

pSOSystem System Calls





Copyright © 1997 Integrated Systems, Inc. All rights reserved. Printed in U.S.A.

Document Title: **pSOSystem System Calls**

Part Number: **000-5070-004**

Revision Date: **August 1997**

Integrated Systems, Inc. • 201 Moffett Park Drive • Sunnyvale, CA 94089-1322

	Corporate	pSOS or pRISM+ Support	MATRIX _x Support
Phone	408-542-1500	1-800-458-7767, 408-542-1925	1-800-958-8885, 408-542-1930
Fax	408-542-1950	408-542-1966	408-542-1951
E-mail	ideas@isi.com	psos_support@isi.com	mx_support@isi.com
Home Page	http://www.isi.com		

LICENSED SOFTWARE - CONFIDENTIAL/PROPRIETARY

This document and the associated software contain information proprietary to Integrated Systems, Inc., or its licensors and may be used only in accordance with the Integrated Systems license agreement under which this package is provided. No part of this document may be copied, reproduced, transmitted, translated, or reduced to any electronic medium or machine-readable form without the prior written consent of Integrated Systems.

Integrated Systems makes no representation with respect to the contents, and assumes no responsibility for any errors that might appear in this document. Integrated Systems specifically disclaims any implied warranties of merchantability or fitness for a particular purpose. This publication and the contents hereof are subject to change without notice.

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS252.227-7013 or its equivalent. Unpublished rights reserved under the copyright laws of the United States.

TRADEMARKS

AutoCode, MATRIX_x, and pSOS are registered trademarks of Integrated Systems, Inc.

The following are trademarks of Integrated Systems, Inc.:

BetterState, BetterState Lite, BetterState Pro, DocumentIt, Epilogue, ESp, HyperBuild, NetState, OpEN, OpTIC, pHILE+, Plug-and-Sim, pNA+, pREPC+, pRISM, pRISM+, pROBE+, pRPC+, pSET, pSOS+, pSOS+m, pSOSim, pSOSystem, pX11+, RealSim, SpOTLIGHT, SystemBuild, Xmath, ZeroCopy.

ARM is a trademark of Advanced RISC Machines Limited. Diab Data, and Diab Data in combination with D-AS, D-C++, D-CC, D-F77, and D-LD are trademarks of Diab Data, Inc. ELANIX, Signal Analysis Module, and SAM are trademarks of ELANIX, Inc. SingleStep is a trademark of Software Development Systems, Inc. SNIFF+ is a trademark of TakeFive Software GmbH, Austria, a wholly-owned subsidiary of Integrated Systems, Inc.

All other products mentioned are the trademarks, service marks, or registered trademarks of their respective holders.



Contents



Contents	iii
Using This Manual	vii
Organization	vii
Conventions	viii
Font Conventions	viii
Symbol Conventions	ix
Format Conventions	ix
Note, Caution, and Warning Conventions	xi
Revision Bar Convention	xii
Commonly Used Terms and Acronyms	xii
Related Publications	xiii
Support	xiv
Contacting Integrated Systems Support	xv

1	pSOS+ System Calls	
2	pHILE+ System Calls	
3	pREPC+ System Calls	
4	pNA+ System Calls	
5	pRPC+ System Calls	
6	pROBE+ and ES_p System Calls	
A	Tables of System Calls	
A.1	Table of All pSOS _{System} Calls	A-1
A.2	pSOS+ System Calls	A-15
A.3	pHILE+ System Calls	A-18
A.4	pREPC+ System Calls	A-20
A.5	pNA+ System Calls	A-26
A.6	pRPC+ System Calls	A-27
A.7	pROBE+ and ES _p System Calls.	A-28
B	Error Codes	
B.1	pSOS+ Error Codes.	B-4
B.2	pHILE+ Error Codes	B-15
B.2.1	pSOS+ Errors Related to pHILE+.	B-34
B.2.2	Conversions of NFS Error Codes	B-35

B.2.3	Conversions of RPC Error Codes	B-36
B.3	pREPC+ Error Codes	B-38
B.4	pNA+ Error Codes	B-39
B.5	pRPC+ Error Codes	B-44
B.6	Driver Error Codes.	B-44
B.6.1	Shared Memory Network Interface Driver Error Codes	B-45
B.6.2	Shared Memory Kernel Interface Driver Error Codes.	B-45
B.6.3	Terminal Interface Driver Error Codes	B-46
B.6.4	Tick Timer Driver Error Codes	B-47
B.6.5	RAM Disk Driver Error Codes.	B-48
B.6.6	TFTP Driver Error Codes	B-48
B.6.7	IDE Driver Error Codes	B-49
B.6.8	FLP Driver Error Codes	B-49
B.6.9	SCSI Driver Error Codes	B-50



Using This Manual



This manual is part of a documentation set that describes pSOSystem, the modular, high-performance real-time operating system environment from Integrated Systems, Inc. This manual is targeted for embedded application developers using the pSOSystem environment. Basic familiarity with UNIX terms and concepts is assumed.

System Calls contains detailed descriptions of all pSOSystem system calls and error codes. For purpose of usability, *System Calls* provides two appendices:

- Appendix A provides an alphabetical list of all system calls with a summary description of each call and a reference to the page where you will find call details. This enables you to search for a call by function when you do not have the specific call name.
- Appendix B provides a numerical list of all error codes returned by pSOSystem. Each error code is listed with its description and the system calls that can return it.

System Calls and other manuals comprise the basic documentation set for the pSOSystem operating system. These other manuals are the *pSOSystem Getting Started*, *pSOSystem System Concepts*, *pSOSystem Programmer's Reference*, *pSOSystem Advanced Topics*, and the *pSOSystem Application Examples*.

Organization

This manual is organized as follows:

Chapter 1, “pSOS+ System Calls,” provides detailed information on each system call in the pSOS+/pSOS+m component of pSOSystem.

Chapter 2, “pHILE+ System Calls,” provides detailed information on each system call in the pHILE+ component of pSOSystem.

Chapter 3, “pREPC+ System Calls,” provides detailed information on each system call in the pREPC+ component of pSOSystem.

Chapter 4, “pNA+ System Calls,” provides detailed information on each system call in the pNA+ component of pSOSystem.

Chapter 5, “pRPC+ System Calls,” provides detailed information on each system call in the pRPC+ component of pSOSystem.

Chapter 6, “pROBE+ and ES_p System Calls,” provides detailed information on the system calls supported by the pROBE+ target debugger/analyzer and the ES_p cross-system visual analyzer.

Appendix A, “Tables of System Calls,” provides a short description of each pSOSystem system call with a reference to the pages that contain detailed information on the call.

Appendix B, “Error Codes,” provides a listing of all error codes returned by pSOSystem system calls.

Conventions

This section describes the conventions used in this manual.

Font Conventions

Fonts other than the standard text default font are used as follows:



<code>Courier</code>	<code>Courier</code> is used for command and function names, file names, directory paths, environment variables, messages and other system output, code and program examples, system calls, and syntax examples.
<code>Courier</code>	User input (anything you are expected to type in) is set in <code>Courier</code> .

<i>italic</i>	<i>Italic</i> is used in conjunction with the default font for emphasis, first instances of terms defined in the glossary, and publication titles.
Bold Helvetica narrow	Buttons, fields, and icons in a graphical user interface are set in bold Helvetica narrow type. Keyboard keys are also set in this type.



Symbol Conventions

This section describes symbol conventions used in this document.

- [] Brackets indicate that the enclosed information is optional. The brackets are generally not typed when the information is entered.
- | A vertical bar separating two text items indicates that either item can be entered as a value.
- The centered dot symbol indicates a required space (for example, in user input).
- % The percent sign indicates the UNIX operating system prompt for C shell.
- \$ The dollar sign indicates the UNIX operating system prompt for Bourne and Korn shells.
-  The symbol of a processor located to the left of text identifies processor-specific information (the example identifies 68K-specific information).
-  Host tool-specific information is identified by a host tools icon (in this example, the text would be specific to the pRISM host tools chain).

Format Conventions

Each reference section in this manual adheres to a standard format. The name of the system call, a brief description, and its C language syntax appear at the top of the first page. The remaining information about the call appears below the syntax and is organized under the following headings:

Volume Types

For pHILE+ system calls only. Names the volume types the system call supports. Volume types include pHILE+, NFS, MS-DOS, and CD-ROM.

Description


Provides a description of the call.

Arguments



Provides descriptions of all arguments used in the call.

Target

Where applicable, provides processor-specific information about the call. The information appears next to an icon representing the processor in question, as below:

 On 68K processors, a signal is passed to the ASR in the D0.L register.

If the information is also specific to a set of host tools, a host tool icon appears next to the processor icon, as below:

  For 68K processors with pRISM host tools, the formula is the following:

$$\text{SIZE} = 32 + (4 * \text{VSIZE}) + (16 * \text{NFD}) + (42 * \text{MAXDEPTH})$$

Return Value

Lists the possible return values of the call. For example, pSOS+ system calls always return a 0 to indicate a successful call, and pREPC+ calls can return either a non-zero value if the test result is true, or 0 if it is false. The Error Codes section lists possible errors returned by each call.

Error Codes

Provides a list of the error codes that the call can generate. For pSOS+ and pHILE+ system calls, an error code returns as a system call *return value*. For other components, such as the pREPC+ component/library and the pNA+ network manager, error codes are loaded into an internal variable that can be read through the macro `errno`. Appendix B contains a complete list of error codes for each software component.

Usage

Provides detailed usage information for certain system calls. For instance, the `verify_vol()` call of the pHILE+ component performs multiple actions that require detailed explanations.

Notes

Provides supplemental information, warnings, and side effects of a call. For pSOS+ system calls, a subsection called “Multiprocessor Considerations” describes the behavior of the call in a multiprocessing environment if it differs from that in a single-processor environment. Also, the subsection “Callable From” lists classes of program elements that the system call can be called from. The system will deadlock if the call is made from a program element not listed in the “Callable From” subsection. There are four possible program elements:

Task — The smallest unit of execution that can compete on its own for system resources.

ISR — Interrupt Service Routine. A function that takes control of the system when the CPU has been triggered with an exception from an external source. An ISR is part of a device driver.

KI — Kernel Interface. The kernel interface is used by pSOS+m to communicate with other pSOS+m kernels on other processors.

Callout — A function that a device driver uses to notify a pSOSSystem component of an interrupt event. A callout is called from an ISR.

See Also

Lists related service calls or the location of other relevant information.

Note, Caution, and Warning Conventions

Within the text of this manual, you may find notes, cautions, and warnings. These statements are used for the purposes described below.

NOTE: Notes provide special considerations or details which are important to the procedures or explanations presented.

CAUTION: **Cautions indicate actions that may result in possible loss of work performed and associated data. An example might be a system crash that results in the loss of data for that given session.**

WARNING: Warnings indicate actions or circumstances that may result in file corruption, irrecoverable data loss, data security risk, or damage to hardware.

Revision Bar Convention

A *revision bar*, at left, appears in the margin next to any text that has changed since the last release of this manual.

Commonly Used Terms and Acronyms

The following terms and acronyms are commonly associated with pSOSSystem and appear in this manual.

ASR	See asynchronous signaloutline.
asynchronous signal routine	A function within an application that executes in response to an asynchronous signal.
callout	A function that a device driver uses to notify a pSOSSystem component of an interrupt event. A callout is called from an ISR.
FD	File descriptor.
FLIST	A contiguous sequence of blocks used to hold file descriptors on a pHILE+ formatted volume.
ISR	See interrupt service routine.
interrupt service routine	A function within an application or device driver that takes control of the system when the CPU has been triggered with an exception from an external source.
KI	See kernel interface.
kernel interface	A user-provided communication layer between nodes in a multiprocessing environment (pSOS+m).
NFS	Network file system.
NI	Network interface.
RSC	See remote service call.
remote service call	A service call made from one node to another in a multiprocessing environment (pSOS+m).

ROOTBLOCK	The root block on a pHILE+ formatted volume, which contains all information needed by pHILE+ to locate other vital information on the volume.
socket	The endpoint for communication across a network.
task	The smallest unit of execution in a system designed with pSOSystem that can compete on its own for system resources.
TCP/IP	Transport Control Protocol/Internet Protocol, a software protocol for communications between computers.
UDP	User Datagram Protocol.

Related Publications

When using the pSOSystem operating system you might want to have on hand the other manuals included in the basic documentation set:

- *pSOSystem Getting Started* explains how to create and bring up pSOSystem-based applications. This manual also contains a number of tutorials.
- *pSOSystem System Concepts* provides theoretical information about the operation of pSOSystem.
- *pSOSystem Programmer's Reference* is the primary source of information on network drivers and interfaces, system services, configuration tables, memory-usage data, and processor-specific assembly languages.
- *pROBE+ User's Manual* describes how to use the pROBE+ target debugger/analyzer.

Based on the options you have purchased, you might also need to refer to one or more of the following manuals:

- *C++ Support Package User's Guide* describes how to implement C++ applications in a pSOSystem environment.
- *SpOTLIGHT Debug Server User's Guide* describes how to use the SpOTLIGHT debugger to debug pSOSystem applications.
- *ESp User's Guide* documents the ESp front-end analyzer, which displays application activities, and the pMONT component, the target-resident application monitor.

- *OpEN User's Guide* describes how to install and use pSOSystem's OPEN (Open Protocol Embedded Networking) product.
- *SNMP User's Guide* describes the internal structure and operation of SNMP, Integrated System's Simple Network Management Protocol product. This manual also describes how to install and use the SNMP MIB (Management Information Base) compiler.

Support

Customers in the United States can contact Integrated Systems Technical Support as described below.

International customers can contact:

- The local Integrated Systems branch office
- The local pSOS distributor
- Integrated Systems Technical Support as described below

Before contacting Integrated Systems Technical Support, please gather the information called for in Table 1 on page xvi. The detailed description in Table 1 should include the following:

- The procedure you followed to build the code. Include components used by the application.
- A complete record of any error messages as seen on the screen (useful for tracking problems by error code).
- A complete test case, if applicable. Attach all include or startup files, as well as a sequence of commands that will reproduce the problem.

Contacting Integrated Systems Support

To contact Integrated Systems Technical Support, use one of the following methods:

- Call 408-980-1500, extension 501 (US and international countries).
- Call 1-800-458-7767 (458-pSOS) (US and international countries with 1-800 support).
- Send a fax to 408-980-1647.
- Send e-mail to psos_support@isi.com.

Integrated Systems actively seeks suggestions and comments about our software, documentation, customer support, and training. Please send your comments by e-mail to ideas@isi.com.

TABLE 1 Problem Report

Contact Name:	
Company Name:	
Customer ID (very important):	
Street Address:	
City, State, Country, Zip Code:	
Voice Phone Number:	
Fax Phone Number:	
E-mail Address:	
Product Name (including components):	
Version(s) :	
Host System:	
Target System:	
Communication Used (ethernet, serial):	
Customer Impact:	
Brief Description:	
Detailed Description (please attach supporting information):	

1

pSOS+ System Calls

This chapter provides detailed information on each system call in the pSOS+ component of pSOSystem. The calls are listed alphabetically, with a multipage section of information for each call. Each call's section includes its syntax, a detailed description, its arguments, its return value, and any error codes that it can return. In addition, it includes information specific to certain processors if such information is needed.

Where applicable, the section also includes the headings "Notes" and "See Also." The "Notes" entry provides important information not specifically related to the call description. In particular, it identifies how the pSOS+m kernel handles the call if multiple processors are involved (see "Multiprocessor Considerations,") and indicates where the call can be made. "See Also" lists other system calls that have related information.

If you need to look up a system call by its functionality, refer to Appendix A, "Tables of System Calls," which lists the calls alphabetically by component and provides a brief description of each call.

For more information on error codes, refer to Appendix B, "Error Codes," which lists the codes numerically and shows which pSOSystem calls are associated with each one.

as_catch Specifies an asynchronous signal routine (ASR).

```
#include <psos.h>
unsigned long as_catch(
    void (* start_addr) (),    /* ASR address */
    unsigned long mode        /* ASR attributes */
)
```

Description

This system call allows a task to specify an asynchronous signal routine (ASR) to handle asynchronous signals. `as_catch()` supplies the starting address of the task's ASR, and its initial execution mode. If the input ASR address is zero, then the caller is deemed to have an invalid ASR, and any signals sent to it will be rejected.

A task's ASR gains control much like an ISR. If a task has pending signals (sent via `as_send()`), then the next time the task is dispatched to run, it will be forced to first execute the task's specified ASR. A task executes its ASR according to the mode supplied by the `as_catch()` call (for example, Non-preemptible, Time-slicing enabled, etc.) Upon entry to the ASR, all pending signals — including all those received since the last ASR invocation — are passed as an argument to the ASR. In addition, a stack frame is built to facilitate the return from the ASR.

`as_catch()` replaces any previous ASR for the calling task. Therefore, a task can have only one ASR at any time. An ASR must exit using the `as_return()` system call.

Arguments

<code>start_addr</code>	Specifies the address of the ASR.
<code>mode</code>	Specifies the ASR's attributes. <code>mode</code> is formed by OR-ing the following symbolic constants (one from each pair), which are defined in <code><psos.h></code> . For instance, to specify that the ASR should have preemption turned off, you place the symbolic constant <code>T_NOPREEMPT</code> in <code>mode</code> . To specify that the ASR should have preemption turned off and roundrobin by time-slicing turned on, you place both <code>T_NOPREEMPT</code> and <code>T_TSLICE</code> in <code>mode</code> , using the following syntax:

```
T_NOPREEMPT | T_TSLICE
```

T_PREEMPT / T_NOPREEMPT	ASR is / is not preemptible.
T_TSLICE / T_NOTSLICE	ASR can / cannot be time-sliced.
T_ASR / T_NOASR	ASR nesting enabled/disabled. If T_ASR is set, then the ASR should be programmed to be re-entrant. If T_NOASR is set, the ASR is prevented from being re-entered as a result of another <code>as_send()</code> call made to that task.
T_USER / T_SUPV	ASR runs in user mode / supervisor mode. See “User and Supervisor Modes” under “Target.”
T_ISR / T_NOISR	Interrupts are enabled / disabled while ASR runs. These options are available only on certain processors. See “Interrupt Control” under “Target.”
T_LEVELMASK0 through T_LEVELMASK _n	Certain interrupts are disabled while ASR runs. These options are available only on certain processors. See “Interrupt Control” under “Target.”

Target

User and Supervisor Modes

You use the symbolic constants T_USER and T_SUPV on each processor as follows:




PPC	On PowerPC, 960 and x86 processors, ASRs execute in supervisor mode only. Hence the symbolic constants T_USER and T_SUPV are ignored.
960	
x86	

Interrupt Control

Interrupt control means that while an ASR is executing, hardware interrupts are disabled. On some processors, you can disable all interrupts at or below a certain interrupt level and enable all interrupts above that level. On other processors you can simply specify that all interrupts are either enabled or disabled. Details are provided below:

How Signals Are Passed to the ASR

The method by which signals are passed to the ASR is processor-specific:

-  On 68K processors, signals are passed to the ASR in the D0.L register.
-  On 960 processors, signals are passed to the ASR in the G0 register.
-  On x86 processors, signals are passed to the ASR in the EAX register.

Return Value

This system call always returns 0.

Error Codes

This system call returns no errors.

Notes

1. An invalid ASR (for example, `start_addr = 0`) should not be confused with the ASR attribute `T_NOASR`. If a task's ASR is invalid, then an `as_send()` call directed to it will be rejected and returned with an error; whereas, the `T_NOASR` attribute simply defers the ASR's execution, with any intervening signals sent to it left pending.
2. A normal task would call `as_catch()` only once, and usually as part of its initialization sequence. Before the first `as_catch()` call, a task is initialized by the pSOS+ kernel to have an invalid ASR.

Multiprocessor Considerations

None. The actions performed by `as_catch()` are entirely confined to the local node, although asynchronous signals can be sent from remote nodes.

Callable From

- Task

See Also

`as_send`, `as_return`

as_return Returns from an asynchronous signal routine (ASR).

```
#include <psos.h>
unsigned long as_return();
```

Description

This system call must be used by a task's ASR to exit and return to the original flow of execution of the task. The purpose of this call is to enable the pSOS+ kernel to restore the task to its state before the ASR. `as_return()` cannot be called except from an ASR.

This call is analogous to the `i_return()` call, which enables an Interrupt Service Routine (ISR) to return to the interrupted flow of execution properly.

Target

Restoring CPU Registers

An ASR is responsible for restoring CPU registers to their previous state before exiting via `as_return()`. The exact way in which this happens varies from processor to processor. On most processors, the ASR is written in assembly language, so you the programmer must take care to restore the registers. On PowerPC processors, an ASR can be written in C, and the pSOS+ kernel restores the registers. Processor-specific information on restoring registers prior to `as_return()` is provided below:

68K

On 68K processors, an ASR is responsible for saving and restoring all CPU registers it uses, including stack pointers. The one exception to this rule is the register D0.L, which is restored by the pSOS+ kernel. On 68K processors, an ASR can be written only in assembly language.

960

On 960 processors, an ASR is responsible for saving and restoring all CPU registers it uses, including stack pointers. The one exception to this rule is the register g0, which is restored by the pSOS+ kernel. On 960 processors, an ASR can be written only in assembly language.

x86

On x86 processors, an ASR is responsible for saving and restoring all CPU registers it uses, including stack pointers. The one exception to this rule is the register EAX, which is restored by the pSOS+ kernel. On x86 processors, an ASR can be written only in assembly language.

Return Value

If successful, this system call never returns. An error code is generated on failure.

Error Codes

Hex	Mnemonic	Description
0x3E	ERR_NOTINASR	Illegal, not called from an ASR.

Notes

Multiprocessor Considerations

None. The actions performed by `as_return()` are confined entirely to the local node.

Callable From

- ASR

See Also

`as_catch`, `as_send`

as_send Sends asynchronous signals to a task.

```
#include <psos.h>
unsigned long as_send(
    unsigned long tid,      /* target task ID */
    unsigned long signals   /* bit-encoded signal list */
)
```

1

Description

This system call sends asynchronous signals to a task. The purpose of these signals is to force a task to break from its normal flow of execution and execute its Asynchronous Signal Routine (ASR).

Asynchronous signals are like software interrupts, with ASRs taking on the role of ISRs. Unlike an interrupt, which is serviced almost immediately, an asynchronous signal does not immediately affect the state of the task. An `as_send()` call is serviced only when the task is next dispatched to run (and that depends on the state of the task and its priority).

Each task has 32 signals. These signals are encoded bit-wise in a single long word. Bits 31 through 16 are reserved for internal system use, and bits 15 through 0 are available for user-specific purposes.

Like events, signals are neither queued nor counted. For example, if three identical signals are sent to a task before its ASR has a chance to execute, the three signals have the same effect as one.

Arguments

<code>tid</code>	Specifies the task to receive the signals.
<code>signals</code>	Contains the bit-encoded signals.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Task has already been deleted.
0x06	ERR_OBJID	tid incorrect, validity check failed.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x3F	ERR_NOASR	Task has no valid ASR.

Notes

1. When an ASR starts execution, all pending asynchronous signals (since its last invocation) are passed to it as an argument.
2. `as_send()` does not trigger the ASR handler if `signals` is 0.

Multiprocessor Considerations

If `tid` identifies a global task that resides on another processor node, the pSOS+ kernel internally makes a remote system call (RSC) to that remote node to send the asynchronous signal to the task.

Callable From

- Task.
- ISR, if the targeted task is local to the node from which the `as_send()` call is made.
- KI, if the targeted task is local to the node from which the `as_send()` call is made.
- Callout, if the targeted task is local to the node from which the `as_send()` call is made.

See Also

`as_catch`

de_close Closes an I/O device.

```
#include <psos.h>
unsigned long de_close(
    unsigned long dev,    /* major/minor device number */
    void *iopb,          /* I/O parameter block address */
    void *retval          /* return value */
)
```

1

Description

The `de_close()` call invokes the device close routine of a pSOS+ device driver specified by the `dev` argument. The functionality of the device close routine is device-specific. For example, an RS-232 device driver close-routine may signal a modem to hang up to signify the end of the connection.

The `de_close()` call, when used in conjunction with `de_open()`, can also be used to implement mutual exclusion. In this case, `de_close()` can be used to signal the end of a critical region for the device operation.

Arguments

<code>dev</code>	Specifies the major and minor device numbers, which are stored in the upper and lower 16 bits, respectively.
<code>iopb</code>	Points to an I/O parameter block, the contents of which are driver-specific.
<code>retval</code>	Points to a variable that receives a driver-specific value returned by the driver.

Return Code

This call returns 0 on success, or an error code on failure. Besides the error codes listed below, other driver-specific errors may be returned.

Error Codes

Hex	Mnemonic	Description
0x101	ERR_IODN	Illegal device (major) number.
0x102	ERR_NODR	No driver provided.
0x103	ERR_IOOP	Illegal I/O function number.

Notes

Callable From

- Task

See Also

de_open

de_cntrl Requests a special I/O device service.

```
#include <psos.h>
unsigned long de_cntrl(
    unsigned long dev,    /* major/minor device number */
    void *iopb,          /* I/O parameter block address */
    void *retval          /* return value */
)
```

1

Description

The `de_cntrl()` call invokes the device control routine of a pSOS+ device driver specified by the `dev` argument. The functionality of a device control routine depends entirely on the device driver implementation. It can include anything that cannot be categorized under the other five I/O services. `de_cntrl()` for a device can be used to perform multiple input and output subfunctions. In such cases, extra parameters in the I/O parameter block can designate the subfunction.

Arguments

<code>dev</code>	Specifies the major and minor device numbers, which are stored in the upper and lower 16 bits, respectively.
<code>iopb</code>	Points to an I/O parameter block, the contents of which are driver-specific.
<code>retval</code>	Points to a variable that receives a driver-specific value returned by the driver.

Return Code

This call returns 0 on success or an error code on failure. Beside the error codes listed below, other driver-specific errors may be returned.

Error Codes

Hex	Mnemonic	Description
0x101	ERR_IODN	Illegal device (major) number.
0x102	ERR_NODR	No driver provided.
0x103	ERR_IOOP	Illegal I/O function number.

Notes

Examples of functions that are often performed by `de_cntrl()` include the following:

- For tty drivers, functions such as changing the baud rate and line-edit characters, enabling/disabling of typed character echo, and so on
- Reading device status information

Callable From

- Task

See Also

`de_read`, `de_write`, `de_open`, `de_close`

de_init

Initializes an I/O device and its driver.

```
#include <psos.h>
unsigned long de_init(
    unsigned long dev,    /* major/minor device number */
    void *iopb,          /* I/O parameter block */
    void *retval,        /* return value */
    void **data_area     /* device data area */
)
```

1

Description

The `de_init()` call invokes the device initialization routine of the pSOS+ device driver specified by the `dev` argument.

The drive init routine can perform one-time device initialization functions such as:

- Resetting the devices
- Setting the necessary programmable registers
- Allocating and/or initializing the driver's data area (for pointers, counters, and so on)
- Creating the messages queues, semaphores, and so on, that are needed for communication and synchronization
- Installing the interrupt vectors, if necessary

Arguments

<code>dev</code>	Specifies the major and minor device numbers, which are stored in the upper and lower 16 bits, respectively.
<code>iopb</code>	Points to an I/O parameter block, the contents of which are driver-specific.
<code>retval</code>	Points to a variable that receives a driver-specific value returned by the driver.

`data_area` This argument is no longer used, but it remains to support compatibility with older drivers and/or pSOS+ application code. The pSOS+ bindings store a value into the variable pointed to by `data_area`. Therefore, a dummy variable still must be allocated to prevent memory corruption.

Return Code

This call returns 0 on success, or an error code on failure. Besides the error codes listed below, other driver-specific errors may be returned.

Error Codes

Hex	Mnemonic	Description
0x101	ERR_IODN	Illegal device (major) number.
0x102	ERR_NODR	No driver provided.
0x103	ERR_IOOP	Illegal I/O function number.

Notes

1. The pSOS+ kernel will automatically call `de_init()` during system initialization if device auto-initialization is enabled. Refer to the *pSOSSystem System Concepts* manual for further details.
2. Normally `de_init()` is called once from the ROOT task for each configured device driver. This call is normally made before other driver services are used.

Callable From

- Task

See Also

`de_open`

de_open Opens an I/O device.

```
#include <psos.h>
unsigned long de_open(
    unsigned long dev,    /* major/minor device number */
    void *iopb,          /* I/O parameter block address */
    void *retval          /* return value */
)
```

1

Description

The `de_open()` call invokes the device open routine of a pSOS+ device driver specified by the `dev` argument.

The device open routine can be used to perform functions that need to be done before the I/O operations can be performed on the device. For example, an asynchronous serial device driver can reset communication parameters (such as baud rate and parity) to a known state for the channel being opened.

A device driver can also assign specific duties to the open routine that are not directly related to data transfer or device operations. For example, a device driver can use `de_open()` to enforce exclusive use of the device during several read and/or write operations.

Arguments

<code>dev</code>	Specifies the major and minor device numbers, which are stored in the upper and lower 16 bits, respectively.
<code>iopb</code>	Points to an I/O parameter block, the contents of which are driver-specific.
<code>retval</code>	Points to a variable that receives a driver-specific value returned by the driver.

Return Code

This call returns 0 on success, or an error code on failure. Besides the error codes listed below, other driver-specific errors may be returned.

Error Codes

Hex	Mnemonic	Description
0x101	ERR_IODN	Illegal device (major) number.
0x102	ERR_NODR	No driver provided.
0x103	ERR_IOOP	Illegal I/O function number.

Notes

Callable From

- Task

See Also

de_close

de_read Reads from an I/O device.

```
#include <psos.h>
unsigned long de_read(
    unsigned long dev,    /* major/minor device number */
    void *iopb,          /* I/O parameter block address */
    void *retval          /* return value */
)
```

1

Description

The `de_read()` call is used to read data from a device. It invokes the device *read* routine of a pSOS+ device driver specified by the `dev` argument. This service normally requires additional parameters contained in the I/O parameter block, such as the address of a data area to hold the data and the number of data units to read.

Arguments

<code>dev</code>	Specifies the major and minor device numbers, which are stored in the upper and lower 16 bits, respectively.
<code>iopb</code>	Points to an I/O parameter block, the contents of which are driver-specific.
<code>retval</code>	Points to a variable that receives a driver-specific value returned by the driver. For example, it can hold the actual number of data units read.

Return Code

This call returns 0 on success, or an error code on failure. In addition to the error codes listed below, other driver-specific errors may be returned.

Error Codes

Hex	Mnemonic	Description
0x101	ERR_IODN	Illegal device (major) number.
0x102	ERR_NODR	No driver provided.
0x103	ERR_IOOP	Illegal I/O function number.

Notes

For many interrupt-driven devices, `de_read()` starts an I/O transaction and blocks the calling task. Most of the I/O transaction can actually be performed in the device's ISR. Upon completion of the transaction, the ISR unblocks the blocked task.

Callable From

- Task

See Also

`de_write`

de_write Writes to an I/O device.

```
#include <psos.h>
unsigned long de_write(
    unsigned long dev,    /* major/minor device number */
    void *iopb,          /* I/O parameter block address */
    void *retval          /* return value */
)
```

1

Description

The `de_write()` call is used to write to a device. It invokes the device write routine of a pSOS+ device driver specified by the `dev` argument. This service normally requires the additional parameters contained in the I/O parameter block, such as the address of the user's output data and the number of data units to write.

Arguments

<code>dev</code>	Specifies the major and minor device numbers, which are stored in the upper and lower 16 bits, respectively.
<code>iopb</code>	Points to an I/O parameter block, the contents of which are driver-specific.
<code>retval</code>	Points to a variable that receives a driver-specific value returned by the driver (the actual number of data units written, for example.)

Return Code

This call returns 0 on success, or an error code on failure. Besides the error codes listed below, other driver-specific errors can be returned.

Error Codes

Hex	Mnemonic	Description
0x101	ERR_IODN	Illegal device (major) number.
0x102	ERR_NODR	No driver provided.
0x103	ERR_IOOP	Illegal I/O function number.

Notes

For many interrupt-driven devices, `de_write()` starts an I/O transaction and blocks the calling task. Most of the I/O transactions can actually be performed in the device's ISR. Upon completion of the transaction, the ISR unblocks the blocked task.

Callable From

- Task

See Also

`de_read`

errno_addr Obtains the address of the calling task's internal `errno` variable.

```
#include <psos.h>
unsigned long *errno_addr();
```

1

Description

This system call returns the address of the calling task's internal `errno` variable.

The pSOS+ kernel maintains an internal `errno` variable for every task. Whenever an error is detected by any pSOSystem component, the associated error code is stored into the running task's internal `errno` variable. The error code can then be retrieved by referencing the `errno` macro defined in the header file `<psos.h>` as follows:

```
#define errno (*(errno_addr()))
```

For example, the following statement expands to include a call to `errno_addr()`:

```
if (errno == ERR_NOMGB)
```

Return Value

This system call returns the address of the `errno` variable of the calling task.

Error Codes

None.

Notes

1. `errno_addr()` provides a unique `errno` value for each task while maintaining compatibility with industry standard library semantics. It should never be necessary to call `errno_addr()` directly from application code.
2. All pSOSystem components set a task's internal `errno` variable. However, for the pSOS+ kernel and pHILE+ file system manager, which return error values via the function return value, use of the `errno` macro is superfluous.
3. A successful system call does not clear the previous `errno` value. `errno` always contains the error code from the last unsuccessful call.

Multiprocessor Considerations

None.

Callable From

- Task

ev_asend (pSOS+m kernel only) Asynchronously sends events to a task.

```
#include <psos.h>
unsigned long ev_asend(
    unsigned long tid,      /* target task identifier */
    unsigned long events    /* bit-encoded events */
)
```

1

Description

This system call asynchronously sends events to a task. It is identical to `ev_send()` except the call is made asynchronously. Refer to the description of `ev_send()` for further information.

Arguments

<code>tid</code>	Specifies the task ID of the target task.
<code>events</code>	Contains a list of bit-encoded events.

Return Value

When called in a system running the pSOS+m kernel, this call always returns 0. The pSOS+ kernel (the single processor version) returns `ERR_SSFN`.

Error Codes

Should the call fail, if present, the node's `MC_ASYNCERR` routine is invoked and the following error codes may be reported:

Hex	Mnemonic	Description
0x05	<code>ERR_OBJDEL</code>	Task has been deleted.
0x06	<code>ERR_OBJID</code>	<code>tid</code> is incorrect, failed validity check.
0x07	<code>ERR_OBJTYPE</code>	Object type doesn't match object ID; failed validity check.

If an `MC_ASYNCERR` routine is not provided, the pSOS+m kernel generates a fatal error.

Notes

1. This call is supported only by the pSOS+m kernel.
2. The events sent to a non-waiting task, or those that do not match the events being waited for, are always left pending.
3. If the `tid` input argument identifies a task residing on the local processor node, the calling task may be preempted as a result of this call.
4. In a multiple-event wait situation, the `ev_send()` and `ev_receive()` pair of calls depend greatly on the temporal course of events. See Note 2 under `ev_receive()` for an example.
5. The pSOS+m kernel does not prevent the use of bits reserved for system use. However, for future compatibility, these bits should not be used.

Multiprocessor Considerations

If the `tid` input argument identifies a global task residing on another processor node, then the pSOS+m kernel will internally make an RSC to that remote node to send the specified `events` to that task.

Callable From

- Task

See Also

`ev_send`, `ev_receive`

ev_receive Enables a task to wait for an event condition.

```
#include <psos.h>
unsigned long ev_receive(
    unsigned long events,    /* bit-encoded events */
    unsigned long flags,    /* event processing attributes */
    unsigned long timeout,  /* timeout delay */
    unsigned long *events_r /* events received */
)
```

1

Description

This service call enables a task to wait for an event condition. The event condition is a set of user-defined events and an ANY/ALL waiting condition qualifier. Each task can wait on 32 events, which are bit-encoded in a long word. An ALL condition occurs when all of the specified events are received. An ANY condition occurs when one or more of the specified events is received.

If the selected event condition is satisfied by events already pending, `ev_receive()` clears those events and returns. Otherwise, `ev_receive()` can return immediately with an error, wait until the requisite events have been received, or wait until a timeout occurs, depending on the `flags` argument.

If successful, `ev_receive()` returns the actual events captured by the call in the location pointed to by `events_r`.

Arguments

events Specifies the set of events. An `events` argument equal to 0 is a special case, where `ev_receive()` returns the pending events but leaves them pending. In this case, the other parameters are ignored.

flags Specifies the event processing attributes. `flags` is formed by OR-ing the following symbolic constants (one from each pair), which are defined in `<psos.h>`. For instance, to specify that `ev_receive()` blocks until all events are satisfied, you place `EV_WAIT` and `EV_ALL` in `flags`, using the following syntax:

```
EV_WAIT | EV_ALL
```

To specify that `ev_receive()` blocks until at least one event is satisfied, you place `EV_WAIT` and `EV_ANY` in `flags`.

	EV_NOWAIT /	Return if the event condition is unsatisfied /
	EV_WAIT	block until the event condition is satisfied.
		Selecting EV_NOWAIT is a convenient way to reset all or selected pending events. For example, an <code>ev_receive()</code> for events 1 and 2 unconditionally resets events 1 and 2.
	EV_ANY /	Wait for ANY / ALL of the desired events.
	EV_ALL	A successful return with EV_ANY signifies that at least one specified event was captured. A successful return with the EV_ALL attribute signifies that all specified events have been captured.
timeout		If EV_WAIT is set, the timeout parameter specifies the timeout in units of clock ticks. If the value of timeout is 0, <code>ev_receive()</code> waits indefinitely. If EV_NOWAIT is set, the timeout argument is ignored.
events_r		Points to the variable where <code>ev_receive()</code> stores the actual events captured.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x01	ERR_TIMEOUT	Timed out; this error code is returned only if EV_WAIT was used and the timeout argument was nonzero.
0x3C	ERR_NOEVS	Selected events not pending; this code is returned only if the EV_NOWAIT attribute was selected.

Notes

1. Events are not accumulated. No matter how many *identical* events are sent to the calling task before it calls `ev_receive()` for receiving the event, the result is the same as if one event were pending.

2. The `ev_receive()` call captures only the events that the caller selects. It captures each selected event once. If a pending event does not match a selected event, the pending event remains pending. Also, if a pending event was sent after an earlier event was used to match a selected event, the pending event remains pending. Consider the following example sequence:
 - a. Task P has pending events 1 and 2.
 - b. With `EV_ALL` set, P calls `ev_receive()` for events 1, 3, and 8. Pending event 1 is cleared.
 - c. Task A sends events 1 and 8 to P.
 - d. Event 1 is made pending. Event 8 is used to match the wanted event.
 - e. Task B sends events 2, 3, and 5 to P. Event 2 has no effect because event 2 is already pending. Event 5 is unwanted and made pending. Event 3 is used to match a wanted event. The event condition is met, so P becomes ready to run.
 - f. Events 1, 2, and 5 are left pending.
 - g. Events 1, 3, and 8 are returned in `events_r`.

Multiprocessor Considerations

None. The actions performed by `ev_receive()` take place only on the local node (whether or not events come from other nodes).

Callable From

- Task

See Also

`ev_send`

ev_send Sends events to a task.

```
#include <psos.h>
unsigned long ev_send(
    unsigned long tid,      /* target task identifier */
    unsigned long events    /* bit-encoded events */
)
```

Description

This system call sends events to a task. If the target task is not waiting for events, the newly sent events are simply made pending. If the task is waiting for events, and the wait condition is fully satisfied as a result of the new events, then the task is unblocked and readied for execution. Otherwise, the task continues to wait. In either case, any of the events sent that do not match those waited on are always left pending.

Each task has 32 events, which are encoded bit-wise in a single long word. Bits 31 through 16 are for internal system use, and bits 15 through 0 are for user-specific purposes. `ev_send()` can send up to 32 different events at one time.

Events are neither queued nor counted. For example, if three identical events are sent to a task before it issues a wait for that event, the three events have the same effect as one event.

Arguments

<code>tid</code>	Specifies the task identifier of the target task.
<code>events</code>	Contains a list of bit-encoded events.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Task has been deleted.
0x06	ERR_OBJID	tid is incorrect, failed validity check.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x65	ERR_STALEID	Object's node has failed.

Notes

1. The events sent to a non-waiting task, or those that do not match the events being waited for, are always simply left pending.
2. If the caller is a task, it may be preempted as a result of this call.
3. In a multiple-event wait situation, the `ev_send()` and `ev_receive()` pair of calls are highly dependent on the temporal course of events. See Note 2 under `ev_receive()` for an example.
4. The pSOS+ kernel does not prevent the use of bits reserved for system use. However, for future compatibility, these bits should not be used.

Multiprocessor Considerations

If the `tid` input argument identifies a global task residing on another processor node, then the pSOS+ kernel internally makes an RSC to that remote node to send the input events to that task.

Callable From

- Task.
- ISR, if the targeted task is local to the node from which the `ev_send()` call is made.
- KI, if the targeted task is local to the node from which the `ev_send()` call is made.
- Callout, if the targeted task is local to the node from which the `ev_send()` call is made.

See Also

`ev_receive`

i_enter

Enters into an interrupt service routine.

This function cannot be called from a high-level language.

Description

This pSOS+ service entry is available only on x86, ColdFire, and PowerPC processors. On these processors, `i_enter()` provides and establishes a standard entrance convention for all interrupt service routines (ISRs). For efficiency reasons, `i_enter()` is different from other system calls in that it uses a separate entry into the pSOS+ kernel.

Target

Entering the ISR



On ColdFire processors, `i_enter()` must be called at the beginning of an ISR so that pSOS+ can keep track of nested interrupts and can switch stacks if an interrupt stack is being used. `i_enter()` is called via TRAP #12.



On PowerPC processors, `i_enter()` must be called at the beginning of an ISR before the interrupt is re-enabled. The `i_enter()` entry point is located at offset 0x60 from the beginning of the pSOS+ kernel code. It's called with a call (bl) instruction.



On x86 processors, `i_enter()` must be used as the first instruction of an ISR, so that pSOS+ can keep track of nested interrupts and can switch stacks if an interrupt stack is being used. `i_enter()` is called via INT 92H.

For more information, please refer to the examples given in the `i_return()` call description on page 1-35.

Return Value

None.

Notes

`i_return()` must be used to exit an ISR.

Multiprocessor Considerations

None. This call can only be directed at the local node.

Callable From

- ISR

i_return Provides an exit from an interrupt service routine.

This call accepts no parameters and cannot be called from a high-level language.

Description

This pSOS+ service entry provides and establishes a standard exit convention for all Interrupt Service Routines (ISRs). It is available for all processors supported by pSOSSystem. For efficiency reasons, `i_return()` is different from other system calls in that it uses a separate, private entry into the pSOS+ kernel. The method of executing an `i_return()` depends on the processor and is explained in “Method of Executing an `i_return()`” under “Target.”

The `i_return()` call is used to integrate ISR level processing within pSOS+. The `i_return()` call detects when all nested ISRs have exited and control is about to be passed back to task level execution. At this transition point, it assesses any task execution state changes that may have taken place during ISR processing, and then passes control to the appropriate task.



Any ISR that makes system calls that may affect the ready state of a task, must conclude with an `i_return()` system call. For those processors which supply an `i_enter()` call, `i_return()` should precede the pSOS+ service calls. Any ISR which contains an `i_enter()` must conclude with an `i_return()`. `i_return()` does not accept any input parameters, and it never returns to the caller.

Before it exits, an ISR must restore CPU registers to their state prior to the interrupt. Processor-specific code examples of ISRs are provided in “Restoring CPU Registers Prior to Exiting the ISR” under “Target.”

Target

Method of Executing an `i_return()`

The method of executing an `i_return()` is processor-specific:

-  On PowerPC processors, `i_return()`'s entry point is located at offset 0x58 from the beginning of the pSOS+ kernel code. It is executed by either a branch (b) or call (bl) instruction.
-  On 960 processors, `i_return()` is executed by a calls 13 instruction.

x86

On x86 processors, `i_return()` is executed by an INT 93H call.

Restoring CPU Registers Prior to Exiting the ISR

The examples below illustrate, for each processor, how an ISR restores CPU registers before exiting via `i_return()`.

CF

On ColdFire processors, an ISR must restore all CPU registers, including the stack pointer, to their state prior to the interrupt. Below is a sample code fragment for an ISR that internally uses CPU registers D0, D1, D2, A0, and A2.

```
TRAP #12
MOVEM.L D0-D2/A0/A2,-(SP)

<body of ISR>

MOVEM.L (SP)+,D0-D2/A0/A2
TRAP #13
```

PPC

On PowerPC processors, right before `i_return()`, the stack pointer must point to the pSOSSystem standard exception frame allocated by the interrupt vector code. Registers R29-R31, LR, CR, SRR0 & SRR1 and SRR2 & SRR3 (for PowerPC 403) have been saved into the frame by the vector code. The values of registers R0, R2-R13, CTR, XER and MQ (for PowerPC 601) prior to the interrupt also need to be saved into the frame by the ISR. `i_return()` will restore all the saved registers and deallocate the frame. Below are sample code fragments of the vector code and an ISR.

Sample vector code:

```
stwu      sp, PS_FRM_SIZE(sp)      # Allocate pSOSSystem
                                           # exception frame
stw       r29, PS_FRM_R29(sp)      # Save r29
stw       r30, PS_FRM_R30(sp)      # Save r30
stw       r31, PS_FRM_R31(sp)      # Save r31
mfsrr0    r29                      #
stw       r29, PS_FRM_SRR0(sp)      # Save SRR0
mfsrr1    r30                      #
stw       r30, PS_FRM_SRR1(sp)      # Save SRR1
mflr      r31                      #
stw       r31, PS_FRM_LR(sp)        # Save LR
mfcrr     r29                      #
stw       r29, PS_FRM_CR(sp)        # Save CR
```

```

andi.    r30, r30, (MSR_IR | MSR_DR | MSR_IR)
mfmsr    r29                                     # Get current MSR
or        r29, r29, r30                         # Restore certain MSR bits
LA        r30, (MSR_FP | MSR_FE1 | MSR_EF0)
or        r29, r29, r30                         # Set certain MSR bits
mtmsr    r29                                     # Set new MSR
isync                                          #
LA        r29, InterruptHandler                # Get interrupt handler
                                                # entry
lwz       r29, 0(r29)                           #
mtlrl     r29                                    #
blr                                              # Jump to handler

```

Sample ISR:

```

LA        r31, pSOSIEnter                      # Get I_ENTER entry
lwz       r31, 0(r31)                           #
mtlrl     r31                                    # Call I_ENTER
blrl                                            #
stw       r0, PS_FRM_R0(sp)                     # Save R0
stw       r2, PS_FRM_R2(sp)                     # Save R2
stw       r3, PS_FRM_R3(sp)                     # Save R3
stw       r4, PS_FRM_R4(sp)                     # Save R4
stw       r5, PS_FRM_R5(sp)                     # Save R5
stw       r6, PS_FRM_R6(sp)                     # Save R6
stw       r7, PS_FRM_R7(sp)                     # Save R7
stw       r8, PS_FRM_R8(sp)                     # Save R8
stw       r9, PS_FRM_R9(sp)                     # Save R9
stw       r10, PS_FRM_R10(sp)                   # Save R10
stw       r11, PS_FRM_R11(sp)                   # Save R11
stw       r12, PS_FRM_R12(sp)                   # Save R12
stw       r13, PS_FRM_R13(sp)                   # Save R13
mfctr     r4                                     # Save CTR
stw       r4, PS_FRM_CTR(sp)                    #
mfixer    r5                                     # Save XER
stw       r5, PS_FRM_XER(sp)                    #
LA        r2, _SDA2_BASE_                       # Set up R2 for ISR
LA        r13, _SDA_BASE_                       # Set up R13 for ISR

<body of ISR>                                   # Handle the interrupt

LA        r31, pSOSIReturn                      # Get I_RETURN entry
lwz       r31, 0(r31)                           #
mtlrl     r31                                    # Jump to I_RETURN and
lr                                              # never return

```

960

On 960 processors, an ISR must restore all global CPU registers, including the frame pointer, to their state prior to the interrupt. Below is a sample code fragment for an ISR that internally uses CPU registers g8, g9, g10, and g11.

```
movq g8,r8
<body of ISR>
movq r8,g8
calls 13
ret
```

x86

On x86 processors, an ISR must restore all CPU registers to their state prior to the interrupt. Below is a sample code fragment for an ISR that internally uses CPU registers ES, EAX, ECX, and EDX.

```
INT      92H                ;PERFORM I_ENTER
PUSH     DS                ;SAVE SOME REGISTERS
PUSH     ES
PUSH     EAX
PUSH     ECX
PUSH     EDX

<body of ISR>              ;HANDLE THE ISR

POP      EDX                ;RESTORE REGISTERS
POP      ECX
POP      EAX
POP      ES
POP      DS
INT      93H                ;PERFORM RETURN
```

SH

On Super Hitachi processors, an ISR must restore all CPU registers, including the stack pointer, to their state prior to the interrupt. Below is a sample code fragment for an ISR that internally uses CPU registers r0, r1, r2, and r3.

```
mov.l    r3, @-sp
mov.l    r2, @-sp
mov.l    r1, @-sp
mov.l    r0, @-sp

<body of ISR>

mov.l    @sp+, r0
mov.l    @sp+, r1
mov.l    @sp+, r2
mov.l    @sp+, r3
trapa    #45
```

Return Value

This system call never returns to the caller.

Notes

`i_return()` should not be used anywhere other than to exit an ISR.

1

Multiprocessor Considerations

None. This call can be directed at the local processor node only.

Callable From

- ISR

k_fatal

Aborts and enters fatal error handling mode.

```
#include <psos.h>
void k_fatal(
    unsigned long err_code,    /* user's error code */
    unsigned long flags       /* fatal condition attributes */
)
```

Description

This system call allows the user application to pass control to the user-defined fatal error handler in the event of a nonrecoverable failure. `k_fatal()` forces a nonrecoverable shutdown of the pSOS+ environment and never returns to the caller.

Arguments

<code>err_code</code>	Specifies a user-defined failure code that is passed to the fatal error handler. The failure code must be at least 0x20000000.
<code>flags</code>	The <code>flags</code> argument is ignored in the single-processor version of the pSOS+ kernel. In a multiprocessor system, the <code>flags</code> argument is used to determine whether the local node should be shut down or a system-wide shutdown should occur. <code>flags</code> is formed by selecting one of the following symbolic constants, which are defined in <code><psos.h></code> (see “Multiprocessor Considerations”).
<code>K_GLOBAL</code> / <code>K_LOCAL</code>	<code>k_fatal()</code> invocation causes global system shutdown / local node shutdown.

If the value of `flags` is `K_GLOBAL`, a global shutdown packet is sent to the master, which then sends a shutdown packet to every other node in the system.

Return Value

This call never returns to the caller.

Notes

1. The shutdown procedure is a procedure whereby pSOS+ attempts to halt execution in the most orderly manner possible. The pSOS+ kernel first examines the pSOS+ Configuration Table entry `kc_fatal`. If this entry is nonzero, the pSOS+ kernel jumps to this address. If `kc_fatal` is zero, and the pROBE+ System Debug/Analyzer is present, then the pSOS+ kernel passes control to the System Failure entry of the pROBE+ debugger. For a description of the pROBE+ debugger behavior in this case, refer to the *pROBE+ User's Manual*. Finally, if the pROBE+ debugger is absent, the pSOS+ kernel internally executes an illegal instruction to cause a deliberate illegal instruction exception. This passes control to a ROM monitor or other low-level debug tool.
2. `k_fatal()` is not the only mechanism by which control is passed to the fatal error handler. It can also receive control following an internal pSOS+ fatal error or, in multiprocessor systems, a shutdown packet from the master node.

Multiprocessor Considerations

In a multiprocessor system, `k_fatal()` can be used to implement a system-wide abort or shutdown. In this case, `K_GLOBAL` should be set. This causes a global shutdown packet to go to the master node, which sends a shutdown packet to every node in the system.

Callable From

- Task
- KI

k_terminate Terminates a node other than the master node.

```
unsigned long k_terminate (  
    unsigned long node,    /* node to terminate */  
    unsigned long fcode,   /* failure code */  
    unsigned long flags    /* unused */  
)
```

Description

This system call enables the user application to shut down a node that it believes has failed or is operating incorrectly. `k_terminate()` causes the specified node to receive a shutdown packet and all other nodes to receive notification of the specified node's failure.

Arguments

node	Specifies the node number of the node to shut down. It cannot be the master node.
fcode	Specifies a user-defined failure code. It must be at least 0x20000000.
flags	Unused.

Return Value

This system returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x04	ERR_NODENO	Node number out of range.
0x67	ERR_MASTER	Cannot terminate master node.

Notes

1. `k_terminate()` can be used to terminate the node from which it is called. In most cases the results are the same as a `k_fatal()` call. However, it is implemented differently. Whereas `k_fatal()` immediately enters the fatal error handler, `k_terminate()` causes a packet to be sent to the master node, which then sends a shutdown packet to the calling node. If the calling node cannot communicate with the master, then the KI presumably calls `k_fatal()` anyway. It is preferable to use `k_fatal()` when the failed node is known to be the local node.
2. A `k_fatal()` call made with the `K_GLOBAL` flag set should be used to shut down the entire system including the master node.

Callable From

- Task
- ISR
- KI
- Callout

See Also

`k_fatal`

m_ext2int Converts an external address into an internal address.

```
#include <psos.h>
unsigned long m_ext2int(
    void *ext_addr,    /* external reference */
    void **int_addr    /* local reference */
)
```

Description

This system call converts an external address into an internal address corresponding to the calling node. A typical use for this conversion is by a node that has received an address from another node that resides in a dual-ported memory zone.

`m_ext2int()` is relevant only to systems with multiple processors connected by dual-ported memory on a memory bus. Other users can disregard this call.

Arguments

<code>ext_addr</code>	Specifies the external address.
<code>int_addr</code>	Points to the variable where <code>m_ext2int()</code> stores the resultant internal address. If the external address is within a dual-ported zone whose p-port is tied to the calling node, then the internal address will be different. In all other cases, the internal and external addresses will be the same.

Return Value

This system call always returns 0.

Error Codes

None.

Notes

1. For descriptions of internal and external addresses and dual-ported memory considerations, see the *pSOSystem System Concepts* manual.
2. Be careful about structures that straddle the boundary of a dual-port zone, because the address range for the structure may contain a discontinuity.

1

Multiprocessor Considerations

None. Although `m_ext2int()` is primarily used in multiprocessor systems, its action is restricted to the local node.

Callable From

- Task
- ISR
- KI
- Callout

See Also

`m_int2ext`

m_int2ext Converts an internal address into an external address.

```
unsigned long m_int2ext(  
    void *int_addr,    /* local reference */  
    void **ext_addr    /* external reference */  
)
```

Description

When a node on a multiprocessor system passes an address that resides within a dual-ported zone, it first must convert the address by calling `m_int2ext()`. This call applies to systems with multiple processors that are connected by dual-ported memory on a memory bus.

Arguments

<code>int_addr</code>	Specifies the internal address.
<code>ext_addr</code>	Points to the variable where <code>m_int2ext()</code> stores the resultant external address. If the internal address is within a dual-ported zone whose p-port is tied to the calling node, the external address is different. In all other cases, the internal and external addresses are the same.

Return Value

This call always returns 0.

Error Codes

None.

Notes

Be careful about structures that straddle the boundary of a dual-port zone, because the structure's address range could contain a discontinuity.

Multiprocessor Considerations

None. Although used in multiprocessor systems, `m_int2ext()` executes on the local node.

Callable From

- Task
- ISR
- KI
- Callout

See Also

`m_ext2int`

pt_create Creates a memory partition of fixed-size buffers.

```
#include <psos.h>
unsigned long pt_create(
    char name[4],           /* partition name */
    void *paddr,            /* partition physical addr. */
    void *laddr,            /* partition logical address */
    unsigned long length,   /* partition length in bytes */
    unsigned long bsize,    /* buffer size in bytes */
    unsigned long flags,    /* buffer attributes */
    unsigned long *ptid,    /* partition identifier */
    unsigned long *nbuf     /* number of buffers created */
)
```

Description

This service call enables a task to create a new memory partition, from which fixed-sized memory buffers can be allocated for use by the application. The pSOS+ kernel takes a portion from the top of this region to use as its Partition Control Block.

Arguments

name	Specifies the user-assigned name for the new partition.
paddr	Specifies the physical memory address of the partition.
laddr	Specifies the logical address of the partition generated after MMU-translation; laddr is ignored on non-MMU systems.
length	Specifies the total partition length in bytes.
bsize	Specifies the size of the buffers. bsize must be a power of 2, and equal to or greater than 4.
flags	Specifies the attributes of the buffer. flags is formed by OR-ing the following symbolic constants (one from each pair), which are defined in <psos.h>. For instance, to specify that a partition is globally addressable, you place the symbolic constant PT_GLOBAL in flags. To specify that the partition is globally addressable and that it prohibits deletion with outstanding buffers, you place both PT_GLOBAL and PT_NODEL in flags, using the following syntax:

```
PT_GLOBAL | PT_NODEL
```


	PT_GLOBAL / PT_LOCAL	Partition is globally addressable by other nodes / partition can be addressed only the by local node. The single-processor version of the pSOS+ kernel ignores PT_GLOBAL.
	PT_DEL / PT_NODEL	Deletion of the partition with pt_delete() is enabled, even if one or more buffers are allocated. Deletion of the partition is prohibited unless all buffers have been freed.
ptid		Points to the variable where pt_create() stores the partition ID of the named partition.
nbuf		Points to the variable where pt_create() stores the number of actual buffers in the partition.

Return Value

This call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x08	ERR_OBJTFULL	Node's object table full.
0x28	ERR_PTADDR	Starting address not on long word boundary.
0x29	ERR_BUFSIZE	Buffer size not power of 2, or less than 4 bytes.
0x2A	ERR_TINYPT	Length too small to hold the partition control information.

Notes

1. Internally, the pSOS+ kernel treats a partition name as a 32-bit integer. However, when the application calls the kernel through the pSOS+ C language API, it passes the partition name as a four-byte character array.
2. The pSOS+ kernel does not check for duplicate partition names. If duplicate names exist, a pt_ident() call can return the ptid of any partition with the duplicate name.

Multiprocessor Considerations

1. The `PT_GLOBAL` attribute should be set only if the partition must be made known to other processor nodes in a multiprocessor configuration. If set, the partition's name and `ptid` are sent to the master node for entry in the Global Object Table.
2. If the `PT_GLOBAL` attribute is set and the number of global objects currently exported by the node equals the Multiprocessor Configuration Table entry `mc_nglbobjs`, then the partition is not created and `ERR_OBJTFULL` is returned.

Callable From

- Task

See Also

`pt_ident`, `pt_getbuf`

pt_delete Deletes a memory partition.

```
#include <psos.h>
unsigned long pt_delete (
    unsigned long ptid    /* partition identifier */
)
```

1

Description

This system call deletes a memory partition specified by its ID. Unless the `PT_DEL` attribute was specified when the partition was created, `pt_delete()` returns an error if any buffers allocated from the partition have not been returned.

The calling task does not have to be the creator (parent) of the partition to be deleted. However, a partition must be deleted from the node on which it was created.

Arguments

`ptid` Specifies the partition identifier.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Partition has been deleted.
0x06	ERR_OBJID	<code>ptid</code> is incorrect, failed validity check.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x2B	ERR_BUFINUSE	Cannot delete; one or more buffers still in use.
0x53	ERR_ILLRSC	Partition not created from this node.

Notes

Once created, a partition is generally used by multiple tasks for data buffers, which can be passed around between tasks, or even between nodes. There is rarely a reason for deleting a partition, even when it is no longer used, except to allow reuse of memory occupied by the partition.

Multiprocessor Considerations

If `ptid` identifies a global partition, `pt_delete` notifies the master node so the partition can be removed from its Global Object Table. Thus, deletion of a global partition always causes activity on the master node.

Callable From

- Task

See Also

`pt_create`

pt_getbuf Gets a buffer from a partition.

```
#include <psos.h>
unsigned long pt_getbuf(
    unsigned long ptid,    /* partition identifier */
    void **bufaddr        /* starting address of buffer */
)
```

1

Description

This system call gets a buffer from a partition. If the partition is empty, an error is returned.

Arguments

- ptid Specifies the partition identifier.
- bufaddr Points to the variable where pt_getbuf() stores the starting address of the allocated buffer.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Partition has been deleted.
0x06	ERR_OBJID	ptid is incorrect, failed validity check.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x2C	ERR_NOBUF	Cannot allocate; partition out of free buffers.
0x65	ERR_STALEID	Object's node has failed.

Notes

1. Buffers always start on long word boundaries.
2. It is not possible to wait for a buffer. `pt_getbuf()` unconditionally returns.

Multiprocessor Considerations

If the input `ptid` identifies a global partition residing on another processor node, then the pSOS+ kernel internally makes an RSC to that remote node to allocate the buffer.

Callable From

- Task.
- ISR, if the partition is local to the node from which `pt_getbuf()` is made.
- KI, if the partition is local to the node from which `pt_getbuf()` is made.
- Callout, if the partition is local to the node from which `pt_getbuf()` is made.

See Also

`pt_retbuf`

pt_ident Obtains the identifier of a named partition.

```
unsigned long pt_ident(  
    char name[4],           /* partition name */  
    unsigned long node,     /* node number */  
    unsigned long *ptid     /* partition identifier */  
)
```

1

Description

This system call enables the calling task to obtain the partition ID of a memory partition it only knows by name. This partition ID can be used in all other operations relating to the memory partition.

Most system calls, except `pt_create()` and `pt_ident()`, reference a partition by its partition ID. `pt_create()` returns the partition ID to the partition creator. For other tasks, one way to obtain the partition ID is to use `pt_ident()`.

Arguments

name	Specifies the name of the partition.
node	For multiprocessing systems, is a search order specifier. See "Multiprocessor Considerations." In a single node system, this argument must be 0.
ptid	Points to the variable where <code>pt_ident()</code> stores the ID of the named partition.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x04	ERR_NODENO	Node specifier out of range.
0x09	ERR_OBJNF	Named partition not found.

Notes

1. Internally, the pSOS+ kernel treats a partition name as a 32-bit integer. However, when the application calls the kernel through the pSOS+ C language API, it passes the partition name as a four-byte character array.
2. The pSOS+ kernel does not check for duplicate partition names. If duplicate partition names exist, a `pt_ident()` call can return the ID of any partition with the duplicate name.

Multiprocessor Considerations

1. `pt_ident()` converts a partition's name to its `ptid` using a search order determined by the `node` input parameter, which is described in *pSOSSystem System Concepts*. Because partitions created and exported by different nodes may not have unique names, the result of this binding may depend on the order in which the object tables are searched.
2. If the master node's Global Object Table must be searched, then the pSOS+m kernel makes an RSC to the master node.

Callable From

- Task

See Also

`pt_create`

pt_retbuf Returns a buffer to the partition from which it came.

```
#include <psos.h>
unsigned long pt_retbuf(
    unsigned long ptid,    /* partition identifier */
    void *bufaddr         /* starting address of the buffer */
)
```

1

Description

This system call returns a buffer to the partition from which it was allocated. Because the pSOS+ kernel does not keep track of buffer ownership, it is possible for one task to get a buffer, and another task to return it.

Arguments

- ptid Specifies the partition ID of the buffer to return.
- bufaddr Specifies the buffer's starting address.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Partition has been deleted.
0x06	ERR_OBJID	ptid is incorrect; failed validity check.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x2D	ERR_BUFADDR	Incorrect buffer starting address.
0x2F	ERR_BUFFFREE	Buffer is already unallocated.
0x65	ERR_STALEID	Object's node has failed.

Notes

Multiprocessor Considerations

If the input `ptid` identifies a global partition residing on another processor node, then the pSOS+ kernel internally makes an RSC to that remote node to return the buffer.

Callable From

- Task.
- ISR, if the partition is local to the node from which the `pt_retbuf()` call is made.
- KI, if the partition is local to the node from which the `pt_retbuf()` call is made.
- Callout, if the partition is local to the node from which the `pt_retbuf()` call is made.

See Also

`pt_getbuf`

pt_sgetbuf Gets a buffer from a partition.

```
#include <psos.h>
unsigned long pt_sgetbuf(
    unsigned long ptid,    /* partition identifier */
    void **paddr,         /* physical address */
    void **laddr          /* logical address */
)
```

1

Description

This system call gets a buffer from a partition. If the partition is empty, an error is returned.

On MMU-based systems, both physical and logical addresses are returned to simplify transfer of buffers between supervisor and user mode programs. In non-MMU systems, the logical address is the same as the physical address, and this call functions the same as the `pt_getbuf()` call.

This service is available in the non-MMU versions of the pSOS+ kernel for the sole purpose of enabling software designed for MMU-based systems to run, unmodified, on systems without MMU.

Arguments

<code>ptid</code>	Specifies the buffer's partition ID.
<code>paddr</code>	Points to the variable where <code>pt_sgetbuf()</code> stores the physical address of the buffer.
<code>laddr</code>	Points to the variable where <code>pt_sgetbuf()</code> stores the logical address of the buffer.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Partition has been deleted.
0x06	ERR_OBJID	ptid is incorrect, failed validity check.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x2C	ERR_NOBUF	Cannot allocate; partition out of free buffers.
0x65	ERR_STALEID	Object's node has failed.

Notes

1. Buffers always start on long word boundaries.
2. It is not possible to wait for a buffer. `pt_sgetbuf()` unconditionally returns.

Multiprocessor Considerations

If the input argument `ptid` identifies a global partition on another processor node, the pSOS+ kernel internally makes an RSC to that remote node to allocate the buffer.

Callable From

- Task

See Also

`pt_retbuf`, `pt_getbuf`

q_asend (pSOS+m kernel only) Asynchronously posts a message to an ordinary message queue.

```
#include <psos.h>
unsigned long q_asend(
    unsigned long qid,          /* queue identifier */
    unsigned long msg_buf[4]    /* message buffer */
)
```

1

Description

This system call functions the same as `q_send()` except that it executes asynchronously. Refer to the description of `q_send()` for further information. For a detailed description of asynchronous services, refer to the *pSOSystem Systems Concepts* manual.

Arguments

- `qid` Specifies the queue ID of the target queue.
- `msg_buf` Specifies the message to send.

Return Value

When called in a system running the pSOS+m kernel this call always returns 0. The pSOS+ kernel (the single processor version) returns `ERR_SSFN`.

Error Codes

Should the call fail, if present, the node's `MC_ASYNCERR` routine is invoked and the following error codes may be reported:

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Queue has been deleted.
0x06	ERR_OBJID	qid incorrect; failed validity check.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x34	ERR_NOMGB	Out of system message buffers.

Hex	Mnemonic	Description
0x35	ERR_QFULL	Message queue at length limit.
0x3A	ERR_VARQ	Queue is variable length
0x65	ERR_STALEID	Object's node has failed.

If an `MC_ASYNCERR` routine is not provided, the pSOS+m kernel generates a fatal error.

Notes

1. This call is supported only by the pSOS+m kernel.
2. The calling task can be preempted as a result of this call.
3. `q_asend()` asynchronously sends a message to an ordinary message queue. Use `q_avsend()` to send a message asynchronously to a variable length message queue.

Multiprocessor Considerations

1. If `qid` identifies a global queue residing on another processor node, then the pSOS+m kernel will internally make an RSC to that remote node to post the input message to that queue.
2. If a task awakened by this call does not reside on the local node, then the pSOS+m kernel will internally pass the message to the task's node of residence, whose pSOS+m kernel will ready the task and give it the relayed message. Thus, a `q_asend()` call, whether it is on the local or a remote queue, may cause pSOS+m activities on another processor node.

Callable From

- Task

See Also

`q_send`, `q_vsend`, `q_avsend`, `q_aurgent`, `q_receive`

q_urgent (pSOS+m kernel only) Asynchronously posts a message at the head of a variable-length message queue.

```
#include <psos.h>
unsigned long q_urgent(
    unsigned long qid,          /* queue identifier */
    unsigned long msg_buf[4]    /* message buffer */
)
```

1

Description

This system call functions the same as the q_urgent() call except that it executes asynchronously. Refer to the description of q_urgent() for further information.

Arguments

- qid Specifies the queue ID of the target queue.
- msg_buf Specifies the message to send.

Return Value

When called in a system running the pSOS+m kernel, this call always returns 0. The pSOS+ kernel (the single processor version) returns ERR_SSFN.

Error Codes

Should the call fail, if present, the node's MC_ASYNCERR routine is invoked and the following error codes can be reported:

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Queue has been deleted.
0x06	ERR_OBJID	qid incorrect; failed validity check.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x34	ERR_NOMGB	Out of system message buffers.
0x35	ERR_QFULL	Message queue at length limit.

Hex	Mnemonic	Description
0x3A	ERR_VARQ	Queue is variable length.
0x65	ERR_STALEID	Object's node has failed.

If an `MC_ASYNCERR` routine is not provided, the pSOS+m kernel generates a fatal error.

Notes

1. This call is supported only by the pSOS+m kernel.
2. The calling task can be preempted as a result of this call.
3. `q_urgent()` asynchronously sends an urgent message to an ordinary message queue. Use `q_avurgent()` to asynchronously send an urgent message to a variable length message queue.

Multiprocessor Considerations

1. If `qid` identifies a global queue residing on another processor node, then the pSOS+m kernel will internally make an RSC to that remote node to post the input message to that queue.
2. If a task awakened by this call does not reside on the local node, then the pSOS+m kernel will internally pass the message to the task's node of residence, whose pSOS+m kernel will ready the task and give it the relayed message. Thus, a `q_urgent()` call, whether it is on the local or a remote queue, may cause pSOS+m activities on another processor node.

Callable From

- Task

See Also

`q_urgent`, `q_vurgent`, `q_avurgent`, `q_asend`, `q_receive`

q_avsend (pSOS+m kernel only) Asynchronously posts a message to a variable-length message queue.

```
#include <psos.h>
unsigned long q_avsend(
    unsigned long qid,          /* queue identifier */
    void *msg_buf,             /* message buffer */
    unsigned long msg_len,      /* length of message */
)
```

1

Description

This system call functions the same as the `q_vsend()` call except that it executes asynchronously. Refer to the description of `q_vsend()` for further information.

Arguments

qid	Specifies the queue ID of the target queue.
msg_buf	Points to the message to send.
msg_len	Specifies the length of the message. It must not exceed the queue's maximum message length.

Return Value

When called in a system running the pSOS+m kernel, this call always returns 0. The pSOS+ kernel (the single processor version) returns `ERR_SSFN`.

Error Codes

Should the call fail, if present, the node's `MC_ASYNCERR` routine is invoked and the following error codes can be reported:

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Queue has been deleted.
0x06	ERR_OBJID	qid incorrect; failed validity check.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.

Hex	Mnemonic	Description
0x31	ERR_MSGSIZ	Message too large.
0x35	ERR_QFULL	Message queue at length limit.
0x3B	ERR_NOTVARQ	Queue is not variable length.
0x65	ERR_STALEID	Object's node has failed.

If an `MC_ASYNCERR` routine is not provided, the pSOS+m kernel generates a fatal error.

Notes

1. This call is supported only by the pSOS+m kernel.
2. The calling task can be preempted as a result of this call.
3. The pSOS+m kernel must copy the message into a queue buffer or the receiving task's buffer. Longer messages take longer to copy. Users should account for the copy time in their designs.
4. `q_avsend()` asynchronously sends a message to a variable length message queue. Use `q_asend()` to send a message asynchronously to an ordinary message queue.

Multiprocessor Considerations

1. If `qid` identifies a global queue residing on another processor node, then the pSOS+m kernel will internally make an RSC to that remote node to post the input message to that queue.
2. If a task awakened by this call does not reside on the local node, the local kernel will internally pass the message to the task's node of residence, whose pSOS+m kernel will ready the task and give it the relayed message. Thus, a `q_avsend()` call, whether it is on the local or a remote queue, may cause pSOS+m activities on another processor node.

Callable From

- Task

See Also

`q_vsend`, `q_send`, `q_asend`, `q_urgent`, `q_vreceive`

q_avurgent (pSOS+m kernel only) Asynchronously posts a message at the head of a variable-length message queue.

```
#include <psos.h>
unsigned long q_avurgent(
    unsigned long qid,          /* queue identifier */
    void *msg_buf,             /* message buffer */
    unsigned long msg_len,      /* length of message */
)
```

Description

This system call functions the same as `q_vurgent` except that `q_avurgent` executes asynchronously. Refer to the description of `q_vurgent` for further information. For a more detailed description of asynchronous services, refer to the *pSOSystem System Concepts* manual.

Arguments

<code>qid</code>	Specifies the queue identifier.
<code>msg_buf</code>	Points to the message to send.
<code>msg_len</code>	Specifies the length of the message.

Return Value

When called in system running pSOS+m, this call always returns 0. The pSOS+ kernel (the single processor version) returns `ERR_SSFN`.

Error Codes

The following error codes may be reported if a `q_avurgent()` call fails and the node's `MC_ASYNCERR` routine (if present) is invoked:

Hex	Mnemonic	Description
0x05	<code>ERR_OBJDEL</code>	Queue has been deleted.
0x06	<code>ERR_OBJID</code>	<code>qid</code> incorrect; failed validity check.
0x07	<code>ERR_OBJTYPE</code>	Object type doesn't match object ID; failed validity check.

Hex	Mnemonic	Description
0x31	ERR_MSGSIZ	Message too large.
0x35	ERR_QFULL	Message queue at length limit.
0x3B	ERR_NOTVARQ	Queue is not variable length.
0x65	ERR_STALEID	Object's node has failed.

If an `MC_ASYNCERR` routine is not present, the pSOS+m kernel generates a fatal error.

Notes

1. This call is supported only by the pSOS+m kernel.
2. The calling task can be preempted as a result of this call.
3. The pSOS+m kernel must copy the message into a queue buffer or the receiving task's buffer. Longer messages take longer to copy. Users should account for the copy time in their designs.
4. `q_avsend()` asynchronously sends a message to a variable length message queue. Use `q_asend()` to asynchronously send a message to an ordinary message queue.

Multiprocessor Considerations

1. If `qid` identifies a global queue residing on another processor node, then the pSOS+m kernel will internally make an RSC to that remote node to post the input message to that queue.
2. If a task awakened by this call does not reside on the local node, the local kernel internally passes the message to the task's node of residence, whose pSOS+m kernel readies the task and gives it the relayed message. Thus, a `q_avurgent()` call, whether it is on the local or a remote queue, can cause pSOS+m activity on another processor node.

Callable From

- Task

See Also

q_urgent, q_vurgent, q_vreceive, q_vsend

q_broadcast Broadcasts identical messages to an ordinary message queue.

```
#include <psos.h>
unsigned long q_broadcast(
    unsigned long qid,          /* queue identifier */
    unsigned long msg_buf[4],  /* msg. of 4 long words */
    unsigned long *count       /* # tasks receiving msg. */
)
```

1

Description

This system call enables the caller to wake up all tasks that might be waiting at an ordinary message queue. If the task queue is empty, this call does nothing. If one or more tasks are waiting at the queue, `q_broadcast()` gives a copy of the input message to each such task and makes it ready to run. After a `q_broadcast()` call, no tasks will be waiting to receive a message from the specified queue.

Arguments

<code>qid</code>	Specifies the queue ID of the target queue.
<code>msg_buf</code>	Specifies the message to send.
<code>count</code>	Points to the variable where <code>q_broadcast()</code> stores the number of tasks readied by the broadcast.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Queue has been deleted.
0x06	ERR_OBJID	<code>qid</code> is incorrect, failed validity check.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x3A	ERR_VARQ	Queue is variable length.

Hex	Mnemonic	Description
0x65	ERR_STALEID	Object's node has failed.

Notes

1. `q_broadcast()` is particularly useful in situations where a single event (for example, an interrupt) must wake up more than one task. In such cases, `q_broadcast()` is clearly more efficient than multiple `q_send()` calls.
2. If the caller is a task, it may be preempted as a result of this call.
3. `q_broadcast()` can be intermixed with `q_send()` and `q_urgent()` calls to the same queue.
4. `q_broadcast()` sends messages to an ordinary message queue. Use `q_vbroadcast()` to send messages to a variable length message queue.

Multiprocessor Considerations

1. If `qid` identifies a global queue residing on another processor node, then the pSOS+m kernel will internally make an RSC to that remote node to post the input message to that queue.
2. If tasks awakened by this call do not reside on the local node, then the pSOS+m kernel will internally pass the message to each task's node of residence, whose pSOS+m kernel will ready the task and give it the relayed message. Thus, a `q_broadcast()` call, whether it is on the local or a remote queue, may cause pSOS+m activities on one or more other processor nodes.

Callable From

- Task.
- ISR, if the targeted queue is local to the node from which the `q_broadcast()` call is made.
- KI, if the targeted queue is local to the node from which the `q_broadcast()` call is made.
- Callout, if the targeted queue is local to the node from which the `q_broadcast()` call is made.

See Also

`q_send`, `q_receive`, `q_vbroadcast`

q_create Creates an ordinary message queue.

```
#include <psos.h>
unsigned long q_create(
    char name[4],           /* queue name */
    unsigned long count,    /* queue size */
    unsigned long flags,    /* queue attributes */
    unsigned long *qid      /* queue identifier */
)
```

Description

This system call creates an ordinary message queue by allocating and initializing a Queue Control Block (QCB) according to the specifications supplied with the call.

Like all objects, a queue has a user-assigned name and a pSOS-assigned queue ID returned by `q_create()`. Several flag bits specify the characteristics of the message queue. Tasks can wait for messages either by task priority or strictly FIFO, and a limit can be optionally set on the maximum number of messages that can be simultaneously posted at the queue.

Arguments

name	Specifies the user-assigned name of the new message queue.
count	If <code>Q_LIMIT</code> is set (see <code>flags</code> , below), then the <code>count</code> argument specifies the maximum number of messages that can be simultaneously posted at the queue. If <code>Q_PRIBUF</code> is also set, then the argument <code>count</code> also specifies the number of buffers set aside from the system-wide pool of message buffers for the private use of this queue. If <code>Q_NOLIMIT</code> is set, <code>count</code> is ignored.
flags	Specifies the attributes of the queue. <code>flags</code> is formed by OR-ing the following symbolic constants (one from each pair), which are defined in <code><psos.h></code> . For instance, to specify that the queue is globally addressable, you place <code>Q_GLOBAL</code> in <code>flags</code> . To specify that the queue is globally addressable and that tasks are queued by FIFO, you place <code>Q_GLOBAL</code> and <code>Q_FIFO</code> in <code>flags</code> , using the following syntax:

```
Q_GLOBAL | Q_FIFO
```

Q_GLOBAL /	Queue is globally addressable by other nodes / queue is
Q_LOCAL	addressable only by the local node.
Q_PRIOR /	Tasks are queued by priority / FIFO.
Q_FIFO	
Q_LIMIT /	Message queue size is limited to count / is unlimited.
Q_NOLIMIT	
Q_PRIBUF /	Private / system buffers are allocated for message
Q_SYSBUF	storage.
qid	Points to the variable where q_create() stores the queue ID of the named queue.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x08	ERR_OBJTFULL	Node's object table full.
0x33	ERR_NOQCB	Cannot allocate QCB; exceeds node's maximum number of active queues.
0x34	ERR_NOMGB	Cannot allocate private buffers; too few available.

Notes

1. Internally, the pSOS+ kernel treats a queue name as a 32-bit integer. However, when the application calls the kernel through the pSOS+ C language API, it passes the queue name as a four-byte character array.
2. The pSOS+ kernel does not check for duplicate queue names. If duplicate names exist, a q_ident() call can return the qid of any queue with the duplicate name.
3. The maximum number of queues that can be simultaneously active is defined by the kc_nqueue entry in the pSOS+ Configuration Table. The count argument is ignored if the Q_NOLIMIT attribute is specified.
4. A queue created with Q_NOLIMIT specified is slightly more efficient.

5. `Q_LIMIT` and a count equal 0 is a legitimate setting. This combination has the interesting property that a `q_send()` will succeed only if there is already a task waiting; otherwise, `q_send()` will fail.
6. `Q_LIMIT` set with `Q_PRIBUF` guarantees that enough buffers will be available for messages to be posted at this queue. If `Q_LIMIT` is not set, then `Q_PRIBUF` is ignored.
7. If a queue is created without private buffers, then messages posted to it will be stored in buffers from the system-wide pool on the node where the queue resides. The size of this pool is defined by the `kc_nmsgbuf` entry in the node's pSOS+ Configuration Table.
8. The `Q_GLOBAL` attribute is ignored by the single-processor version of the pSOS+ kernel.
9. `q_create()` creates an ordinary message queue. Use `q_vcreate()` to create a variable length message queue.

Multiprocessor Considerations

1. The `Q_GLOBAL` attribute should be set only if the queue must be made known to other processor nodes in a multiprocessor configuration. If set, the queue's name and `qid` are sent to the master node for entry in its Global Object Table.
2. If the `Q_GLOBAL` attribute is set and the number of global objects currently exported by the node equals the Multiprocessor Configuration Table entry `mc_nglbobj`, then the queue is not created and `ERR_OBJTFULL` is returned.

Callable From

- Task

See Also

`q_ident`, `q_delete`, `q_vcreate`

q_delete Deletes an ordinary message queue.

```
#include <psos.h>
unsigned long q_delete(
    unsigned long qid    /* queue identifier */
)
```

1

Description

This system call deletes the ordinary message queue with the specified queue ID, and frees the QCB. `q_delete()` takes care of cleaning up the queue. If there are tasks waiting, they will be unblocked and given an error code. If some messages are queued there, the message buffers, along with any free private buffers are returned to the system-wide pool.

The calling task does not have to be the creator of the queue in order to be deleted. However, a queue must be deleted from the node on which it was created.

Arguments

`qid` Specifies the queue ID of the queue to delete.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Queue has been deleted.
0x06	ERR_OBJID	qid is incorrect; failed validity check.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x38	ERR_TATQDEL	Informative only; there were tasks waiting at the queue.
0x39	ERR_MATQDEL	Informative only; there were messages pending in the queue.

Hex	Mnemonic	Description
0x3A	ERR_VARQ	Queue is variable length.
0x53	ERR_ILLRSC	Queue not created from this node.

Notes

1. Once created, a queue is generally used by multiple tasks for communication and synchronization. There is rarely a reason for deleting a queue, even when it is no longer used, except to allow reuse of the QCB.
2. The calling task may be preempted after this call, if a task that is waiting for a message from the deleted queue has higher priority.
3. Any pending messages are lost.
4. `q_delete()` deletes an ordinary message queue. Use `q_vdelete()` to delete a variable length message queue.

Multiprocessor Considerations

If `qid` identifies a global queue, `q_delete` will notify the master node so that the queue can be removed from its Global Object Table. Thus, deletion of a global queue always causes activity on the master node.

Callable From

- Task

See Also

`q_create`, `q_vdelete`

q_ident Obtains the queue ID of an ordinary message queue.

```
#include <psos.h>
unsigned long q_ident(
    char name[4],          /* queue name */
    unsigned long node,    /* node number */
    unsigned long *qid      /* queue identifier */
)
```

1

Description

The intended purpose of this system call is to enable the calling task to obtain the queue ID of an ordinary message queue. However, since a variable length message queue is just a special type of message queue, `q_ident()` and `q_vident()` are functionally identical. Both return the queue ID of the first queue encountered with the specified name, whether it be ordinary or variable length.

Most system calls, except `q_create()/q_vcreate()` and `q_ident()/q_vident()`, reference a queue by its queue ID. For other tasks, one way to obtain the queue ID is to use `q_ident()/q_vident()`. Once obtained, the queue ID can then be used in all other operations relating to this queue.

Arguments

name	Specifies the name of the message queue.
node	For multiprocessing systems, is a search order specifier. See "Multiprocessor Considerations." In a single node system, this argument must be 0.
qid	Points to the variable where <code>q_ident()</code> stores the ID of the named message queue.

Return Value

The system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x04	ERR_NODENO	Node specifier out of range.
0x09	ERR_OBJJNF	Named queue not found.

Notes

1. Internally, the pSOS+ kernel treats a queue name as a 32-bit integer. However, when the application calls the kernel through the pSOS+ C language API, it passes the queue name as a four-byte character array.
2. The pSOS+ kernel does not check for duplicate queue names. If duplicate names exist, a `q_ident()` call can return the `qid` of any queue with the duplicate name.

Multiprocessor Considerations

1. `q_ident()` converts a queue's name to its `qid` using a search order determined by the `node` input parameter, as described in *pSOSystem System Concepts*. Because queues created and exported by different nodes may not have unique names, the result of this binding may depend on the order in which the object tables are searched.
2. If the master node's Global Object Table must be searched, the local kernel makes a `q_ident()` RSC to the master node.

Callable From

- Task

See Also

`q_create`, `q_vident`

q_receive Requests a message from an ordinary message queue.

```
#include <psos.h>
unsigned long q_receive(
    unsigned long qid,          /* queue identifier */
    unsigned long flags,        /* queue attributes */
    unsigned long timeout,      /* timeout in clock ticks */
    unsigned long msg_buf[4]    /* message buffer */
)
```

1

Description

This system call enables a task or an ISR to obtain a message from an ordinary message queue.

Arguments

qid	Specifies the queue ID of the target queue.				
flags	Specifies whether <code>q_receive()</code> will block waiting for a message. <code>flags</code> should have one of the following values (defined in <code><psos.h></code>): <table border="0" style="margin-left: 20px;"> <tr> <td><code>Q_NOWAIT</code></td><td>Don't wait for message.</td></tr> <tr> <td><code>Q_WAIT</code></td><td>Wait for message.</td></tr> </table>	<code>Q_NOWAIT</code>	Don't wait for message.	<code>Q_WAIT</code>	Wait for message.
<code>Q_NOWAIT</code>	Don't wait for message.				
<code>Q_WAIT</code>	Wait for message.				
timeout	Specifies the timeout interval, in units of clock ticks.				
msg_buf	An output parameter. Contains the received message.				

If the queue is non-empty, this call always returns the first message there. If the queue is empty and the caller specified `Q_NOWAIT`, then `q_receive()` returns with an error code. If `Q_WAIT` is elected, the caller will be blocked until a message is posted to the queue, or if the `timeout` argument is used, until the timeout occurs whichever happens first. If `timeout` is zero and `Q_WAIT` is selected, then `q_receive()` will wait forever. The `timeout` argument is ignored if `Q_NOWAIT` is selected.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x01	ERR_TIMEOUT	Timed out; this error code is returned only if a timeout was requested.
0x05	ERR_OBJDEL	Queue has been deleted.
0x06	ERR_OBJID	qid incorrect; failed validity checks.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x36	ERR_QKILLD	Queue deleted while task waiting.
0x37	ERR_NOMSG	Queue empty; this error code is returned only if Q_NOWAIT was selected.
0x54	ERR_NOAGNT	Cannot wait; the remote node is out of agents.
0x3A	ERR_VARQ	Queue is variable length.
0x65	ERR_STALEID	Object's node has failed.
0x66	ERR_NDKLDD	Object's node failed while RSC waiting.

Notes

1. If it is necessary to block the calling task, `q_receive()` will enter the calling task at the message queue's task-wait queue. If the queue was created with the `Q_FIFO` attribute, then the caller is simply entered at the tail of the wait queue. If the queue was created with the `Q_PRIOR` attribute, then the task will be inserted into the wait queue by priority.
2. `q_receive()` requests a message from an ordinary message queue. Use `q_vreceive()` to request a message from a variable length message queue.

Multiprocessor Considerations

If `qid` identifies a global queue residing on another processor node, the local kernel will internally make an RSC to that remote node to request a message from that queue. If the `Q_WAIT` attribute is elected, then the pSOS+m kernel on the target node must use an *agent* to wait for the message. An agent is an internal object created by pSOS+ to simulate a task on a remote node. If the node is temporarily

q_send

Posts a message to an ordinary message queue.

```
#include <psos.h>
unsigned long q_send(
    unsigned long qid,          /* queue identifier */
    unsigned long msg_buf[4]    /* message buffer */
)
```

Description

This system call is used to send a message to a specified ordinary message queue. If a task is already waiting at the queue, the message is passed to that task, which is then unblocked and made ready to run. If no task is waiting, the input message is copied into a message buffer from the system pool or, if the queue has private buffers, into a private message buffer, which is then put in the message queue behind any messages already posted to the queue.

Arguments

qid	Specifies the queue ID of the target queue.
msg_buf	Specifies the message to send.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Queue has been deleted.
0x06	ERR_OBJID	qid incorrect; failed validity check.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x34	ERR_NOMGB	Out of system message buffers.
0x35	ERR_QFULL	Message queue at length limit.

Hex	Mnemonic	Description
0x3A	ERR_VARQ	Queue is variable length.
0x65	ERR_STALEID	Object's node has failed.

Notes

1

1. If the caller is a task, it may be preempted as a result of this call.
2. `q_send()` sends a message to an ordinary message queue. Use `q_vsend()` to send a message to a variable length message queue.

Multiprocessor Considerations

1. If `qid` identifies a global queue residing on another processor node, the local kernel will internally make an RSC to that remote node to post the input message to that queue.
2. If a task awakened by this call does not reside on the local node, the local kernel will internally pass the message to the task's node of residence, whose pSOS+m kernel will ready the task and give it the relayed message. Thus, a `q_send()` call, whether it is on the local or a remote queue, may cause pSOS+m activities on another processor node.

Callable From

- Task.
- ISR, if the target queue is local to the node from which the `q_send()` call is made.
- KI, if the target queue is local to the node from which the `q_send()` call is made.
- Callout, if the target queue is local to the node from which the `q_send()` call is made.

See Also

`q_broadcast`, `q_receive`, `q_urgent`, `q_vsend`

q_urgent Posts a message at the head of an ordinary message queue.

```
#include <psos.h>
unsigned long q_urgent(
    unsigned long qid,          /* queue identifier */
    unsigned long msg_buf[4]    /* message buffer */
)
```

Description

This system call is identical in all respects to `q_send()` with one exception: if one or more messages are already posted at the target queue, then the new message will be inserted into the message queue in front of all such queued messages.

Arguments

<code>qid</code>	Specifies the queue ID of the target queue.
<code>msg_buf</code>	Specifies the message to send.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Queue has been deleted.
0x06	ERR_OBJID	<code>qid</code> incorrect; failed validity check.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x34	ERR_NOMGB	Out of system message buffers.
0x35	ERR_QFULL	Message queue at length limit.
0x3A	ERR_VARQ	Queue is variable length.
0x65	ERR_STALEID	Object's node has failed.

Notes

1. `q_urgent()` is useful when the message represents an urgent errand and must be serviced ahead of the normally FIFO ordered messages.
2. If the caller is a task, it may be preempted as a result of this call.
3. `q_urgent()` sends a message to an ordinary message queue. Use `q_vurgent()` to send a message to a variable length message queue.

1

Multiprocessor Considerations

1. If `qid` identifies a global queue residing on another processor node, the local kernel internally makes an RSC to that remote node to post the input message to that queue.
2. If a task awakened by this call does not reside on the local node, the local kernel internally passes the message to the task's node of residence, whose pSOS+m kernel will ready the task and give it the relayed message. Thus, a `q_urgent()` call, whether it is on the local or a remote queue, may cause pSOS+m activities on another processor node.

Callable From

- Task.
- ISR, if the target queue is local to the node from which the `q_urgent()` call is made.
- KI, if the target queue is local to the node from which the `q_urgent()` call is made.
- Callout, if the target queue is local to the node from which the `q_urgent()` call is made.

See Also

`q_receive`, `q_send`, `q_vurgent`

q_vbroadcast Broadcasts identical variable-length messages to a message queue.

```
#include <psos.h>
unsigned long q_vbroadcast(
    unsigned long qid,      /* queue identifier */
    void *msg_buf,         /* message buffer */
    unsigned long msg_len, /* length of message */
    unsigned long *count   /* number of tasks */
)
```

Description

This system call sends a message to all tasks waiting at a specified variable length queue. Otherwise, it is identical to `q_broadcast()`.

Arguments

<code>qid</code>	Specifies the queue ID of the target queue.
<code>msg_buf</code>	Points to the message to send.
<code>msg_len</code>	Specifies the length of the message. It must not exceed the queue's maximum message length, which was specified with <code>q_vcreate()</code> .
<code>count</code>	Points to the variable where <code>q_vbroadcast()</code> stores the number of tasks readied by the broadcast.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Queue has been deleted.
0x06	ERR_OBJID	<code>qid</code> is incorrect, failed validity check.
0s07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.

Hex	Mnemonic	Description
0x30	ERR_KISIZE	Message buffer length exceeds max. KI packet buffer length.
0x31	ERR_MSGSIZ	Message is too large.
0x3B	ERR_NOTVARQ	Queue is not variable length.
0x65	ERR_STALEID	Object's node has failed.

Notes

1. If the caller is a task, it may be preempted as a result of this call.
2. `q_vbroadcast()` can be intermixed with `q_vsend()` and `q_vurgent()` calls to the same queue.
3. The pSOS+ kernel must copy the message from the caller's buffer to a receiving task's buffer. Longer messages take longer to copy. Users should account for the copy time in their designs, especially when calling from an ISR.
4. `q_vbroadcast()` sends messages to a variable length message queue. Use `q_broadcast()` to send messages to an ordinary queue.

Multiprocessor Considerations

1. If `qid` identifies a global queue residing on another processor node, the local kernel will internally make an RSC to that remote node to post the input message to that queue.
2. If tasks awakened by this call do not reside on the local node, the local kernel internally passes the message to each task's node of residence, whose pSOS+ kernel will ready the task and give it the relayed message. Thus, a `q_vbroadcast()` call, whether it is on the local or a remote queue, may cause pSOS+m activities on one or more processor nodes.

Callable From

- Task.
- ISR, if the target queue is local to the node from which the `q_vbroadcast()` call is made.

See Also

`q_broadcast`, `q_vsend`, `q_vreceive`

q_vcreate Creates a variable-length message queue.

```
#include <psos.h>
unsigned long q_vcreate(
    char name[4],           /* queue name */
    unsigned long flags,    /* queue characteristics */
    unsigned long maxnum,   /* maximum number of messages */
    unsigned long maxlen,   /* maximum message length */
    unsigned long *qid      /* queue identifier */
)
```

1

Description

This system call creates a queue that supports variable length messages. Otherwise, it is identical to `q_create()`. `q_vcreate` creates a message queue by allocating and initializing a Queue Control Block (QCB) according to the specifications supplied with the call.

Arguments

<code>name</code>	Specifies the user-assigned name of the new message queue.
<code>flags</code>	Specifies the attributes of the queue. <code>flags</code> is formed by OR-ing the following symbolic constants (one from each pair), which are defined in <code><psos.h></code> . For instance, to specify that the queue is globally addressable, you place <code>Q_GLOBAL</code> in <code>flags</code> . To specify that the queue is globally addressable and that tasks are queued by FIFO, you place <code>Q_GLOBAL</code> and <code>Q_FIFO</code> in <code>flags</code> , using the following syntax: <div style="margin-left: 40px;"> <code>Q_GLOBAL Q_FIFO</code> <code>Q_GLOBAL / Q_LOCAL</code> Queue is globally addressable by other nodes / queue is addressable only by the local node. <code>Q_PRIOR / Q_FIFO</code> Tasks are queued by priority / FIFO. </div>
<code>maxnum</code>	Specifies the maximum number of messages that can be pending at one time at the queue.
<code>maxlen</code>	Specifies the maximum message size (in bytes).

qid Points to the variable where `q_vcreate()` stores the queue's pSOS-assigned queue ID.

Queues created by `q_vcreate()` always have a fixed number of private buffers. The pSOS+ kernel uses `maxnum` and `maxlen` to allocate sufficient memory for message storage from region 0. If insufficient memory is available, an error is returned. Queues created with `q_vcreate()` never allocate or use message buffers from the pSOS+ message buffer pool.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x08	ERR_OBJSFULL	Node's object table full.
0x30	ERR_KISIZE	Global queue maxlen too large for KI.
0x33	ERR_NOQCB	Cannot allocate QCB; exceeds node's maximum number of active queues.
0x34	ERR_NOMGB	Not enough memory in region 0.

Notes

1. Internally, the pSOS+ kernel treats a queue name as a 32-bit integer. However, when the application calls the kernel through the pSOS+ C language API, it passes the queue name as a four-byte character array.
2. The pSOS+ kernel does not check for duplicate queue names. If duplicate names exist, an `q_vident()` can return the `qid` of any queue with the duplicate name.
3. The maximum number of queues that can be simultaneously active is defined by the entry `kc_nqcb` in the pSOS+ Configuration Table. It applies to the combined total of both fixed and variable queues.
4. The `Q_GLOBAL` attribute is ignored by the single-processor version of the pSOS+ kernel.
5. A special case occurs when `maxnum` is set to 0. In this case, a message can only be successfully sent if a task is already waiting at the queue.

6. A special case occurs when `maxlen` is set to 0. In this case, the queue behaves much like a counting semaphore.
7. No memory is allocated by the queue when either `maxlen` or `maxnum` is set to 0. The amount of Region 0 memory needed by the queue is given by the formula in the section of this manual called "Memory Usage."
8. `q_vcreate()` creates a variable length message queue. Use `q_create()` to create an ordinary queue.

1

Multiprocessor Considerations

1. The `Q_GLOBAL` attribute should be set only if the queue must be made known to other processor nodes in a multiprocessor configuration. If set, the queue's name and `qid` are sent to the master node for entry in its Global Object Table.
2. If the `Q_GLOBAL` attribute is set and the number of global objects currently exported by the node equals the Multiprocessor Configuration Table entry `mc_nglbobj` then the queue is not created and `ERR_OBJTFULL` is returned.
3. If the maximum message length as specified by `maxlen` might require transmission of a packet larger than the KI can transmit, as specified in the Multiprocessor Configuration Table entry `mc_kimaxbuf`, then the queue is not created and `ERR_KISIZE` is returned.

Callable From

- Task

See Also

`q_create`, `q_vident`, `q_vdelete`

q_vdelete

Deletes a variable-length message queue.

```
#include <psos.h>
unsigned long q_vdelete(
    unsigned long qid    /* queue identifier */
)
```

Description

This system call deletes a variable length message queue. Otherwise, it is identical to `q_delete()`.

Arguments

`qid` Specifies the queue ID of the queue to delete.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Queue has been deleted.
0x06	ERR_OBJID	qid is incorrect; failed validity check.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x38	ERR_TATQDEL	Informative only; tasks were waiting at the queue.
0x39	ERR_MATQDEL	Informative only; messages were pending in the queue.
0x3B	ERR_NOTVARQ	Queue is not variable length.
0x53	ERR_ILLRSC	Queue not created from this node.

Notes

1. Message storage is returned to region 0. Hence the calling task can be preempted by a high priority task waiting for memory.
2. The calling task can also be preempted after this call, if a task waiting at the deleted queue has higher priority.
3. Any pending messages are lost.
4. `q_vdelete()` deletes a variable length message queue. Use `q_delete()` to delete an ordinary queue.

Multiprocessor Considerations

If `qid` identifies a global queue, `q_vdelete()` will notify the master node so that the queue can be removed from its Global Object Table. Thus, deletion of a global queue always causes activity on the master node.

Callable From

- Task

See Also

`q_delete`, `q_vcreate`

q_vident Obtains the queue ID of a variable-length message queue.

```
#include <psos.h>
unsigned long q_vident(
    char name[4],          /* queue name */
    unsigned long node,    /* node number */
    unsigned long *qid     /* queue identifier */
)
```

Description

The intended purpose of this system call is to allow the calling task to obtain the queue ID of a variable length message queue. However, since a variable length message queue is just a special type of message queue, `q_ident()` and `q_vident()` are functionally identical. Both return the queue ID of the first variable length or fixed length queue encountered with the specified name.

Arguments

name	Specifies the user-assigned name of the message queue.
node	For multiprocessor systems, is a search order specifier. See “Multiprocessor Considerations.” In a single node system, this argument must be 0.
qid	Points to the variable where <code>q_vident()</code> stores the queue ID of the named queue.

Return Value

The system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x04	ERR_NODENO	Node specifier out of range.
0x09	ERR_OBJNFB	Named queue not found.

Notes

1. Internally, the pSOS+ kernel treats a queue name as a 32-bit integer. However, when the application calls the kernel through the pSOS+ C language API, it passes the queue name as a four-byte character array.
2. The pSOS+ kernel does not check for duplicate queue names. If duplicate names exist, a `q_vident()` call can return the `qid` of any queue with the duplicate name.

Multiprocessor Considerations

1. `q_vident()` converts a queue's name to its `qid` using a search order determined by the `node` input parameter as described in *pSOSystem System Concepts*. Because queues created and exported by different nodes may not have unique names, the result of this binding may depend on the order in which the object tables are searched.
2. If the master node's Global Object Table must be searched, the local kernel makes an `q_vident()` RSC to the master node.

Callable From

- Task

See Also

`q_ident`, `q_vcreate`

q_vreceive Requests a message from a variable-length message queue.

```
#include <psos.h>
unsigned long q_vreceive(
    unsigned long qid,          /* queue identifier */
    unsigned long flags,        /* queue attributes */
    unsigned long timeout,      /* timeout in clock ticks */
    void *msg_buf,              /* message buffer */
    unsigned long buf_len,      /* length of buffer */
    unsigned long *msg_len      /* length of message */
)
```

Description

This system call enables a task or an ISR to obtain a message from a *variable length* message queue. Otherwise, it is identical to `q_receive()`.

Arguments

qid	Specifies the queue ID of the target queue.
flags	Specifies whether <code>q_vreceive()</code> will block waiting for a message. <code>flags</code> should have one of the following values (defined in <code><psos.h></code>): <div style="margin-left: 20px;"> <code>Q_NOWAIT</code> / Don't wait / wait for message.. <code>Q_WAIT</code> </div>
timeout	Specifies the timeout interval, in units of clock ticks. If <code>timeout</code> is zero and <code>Q_WAIT</code> is selected, then <code>q_vreceive()</code> will wait forever. <code>timeout</code> will be ignored if <code>Q_NOWAIT</code> is selected.
msg_buf	Points to the buffer that receives the message.
buf_len	Specifies the length of the buffer <code>msg_buf</code> points to. If <code>buf_len</code> is less than the queue's maximum message length, <code>ERR_BUFSIZ</code> is returned to the caller.
msg_len	Points to the variable where <code>q_receive()</code> stores the actual length of the received message.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x01	ERR_TIMEOUT	Timed out: this error code is returned only if a timeout was requested.
0x05	ERR_OBJDEL	Queue has been deleted.
0x06	ERR_OBJID	qid incorrect; failed validity checks.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x30	ERR_KISIZE	Message buffer length exceeds max. KI packet buffer length.
0x32	ERR_BUFSIZ	Buffer is too small.
0x36	ERR_QKILLD	Queue deleted while task waiting.
0x37	ERR_NOMSG	Queue empty; this error code is returned only if Q_NOWAIT was selected.
0x3B	ERR_NOTVARQ	Queue is not variable length.
0x54	ERR_NOAGNT	Cannot wait; the remote node is out of agents.
0x65	ERR_STALEID	Object's node has failed.
0x66	ERR_NDKLD	Object's node failed while remote service call (RSC) waiting.

Notes

1. If it is necessary to block the calling task, q_vreceive() will enter the calling task in the queue's task-wait queue. If the queue was created with the Q_FIFO attribute, then the caller is simply entered at the tail of the wait queue. If the

queue was created with the `Q_PRIOR` attribute, then the task will be inserted into the wait queue by priority.

2. The pSOS+ kernel must copy the message from the queue into the caller's buffer. Longer messages take longer to copy. User's should account for the copy time in their design, especially when calling from an ISR.
3. `q_vreceive()` requests a message from a variable length message queue. Use `q_receive()` to request a message from an ordinary queue.

Multiprocessor Considerations

If `qid` identifies a global queue residing on another processor node, the local kernel will internally make an RSC to that remote node to request a message from that queue. If the `Q_WAIT` attribute is elected, then the pSOS+m kernel on the target node must use an agent to wait for the message. If that node is temporarily out of agents, the call will fail. The number of agents on each node is defined by the `mc_nagent` entry in the Multiprocessor Configuration Table.

Callable From

- Task.
- ISR, if `Q_NOWAIT` is set.
- KI, if `Q_NOWAIT` is set.
- Callout, if `Q_NOWAIT` is set.

See Also

`q_receive`, `q_vsend`

q_vsend

Posts a message to a specified variable-length message queue.

```
#include <psos.h>
unsigned long q_vsend(
    unsigned long qid,      /* queue identifier */
    void *msg_buf,         /* message buffer */
    unsigned long msg_len, /* length of message */
)
```

1

Description

This system call is used to send a message to a specified variable length message queue. Other than the queue type, `q_vsend()` operates just like `q_send()`.

Arguments

qid	Specifies the queue ID of the target queue.
msg_buf	Points to the message to send.
msg_len	Specifies the length of the message. It must not exceed the queue's maximum message length.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Queue has been deleted.
0x06	ERR_OBJID	qid incorrect; failed validity check.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x30	ERR_KISIZE	Message buffer length exceeds max. KI packet buffer length.
0x31	ERR_MSGSIZ	Message is too large.
0x35	ERR_QFULL	Message queue at length limit.

Hex	Mnemonic	Description
0x3B	ERR_NOTVARQ	Queue is not variable length.
0x65	ERR_STALEID	Object's node has failed.

Notes

1. If the caller is a task, it may be preempted as a result of this call.
2. The pSOS+ kernel must copy the message into a queue buffer or the receiving task's buffer. Longer messages take longer to copy. User's should account for the copy time in their design, especially when calling from an ISR.
3. `q_vsend()` sends a message to a variable length message queue. Use `q_send()` to send a message to an ordinary queue.

Multiprocessor Considerations

1. If `qid` identifies a global queue residing on another processor node, the local kernel will internally make an RSC to that remote node to post the input message to that queue.
2. If a task awakened by this call does not reside on the local node, the local kernel will internally pass the message to the task's node of residence, whose pSOS+m kernel will ready the task and give it the relayed message. Thus, a `q_vsend()` call, whether it is on the local or a remote queue, may cause pSOS+m activities on another processor node.

Callable From

- Task.
- ISR, if the target queue is local to the node from which the `q_vsend()` call is made.
- KI, if the target queue is local to the node from which the `q_vsend()` call is made.
- Callout, if the target queue is local to the node from which the `q_vsend()` call is made.

See Also

`q_send`, `q_vreceive`

q_vurgent Posts a message at the head of a variable-length message queue.

```
#include <psos.h>
unsigned long q_vurgent(
    unsigned long qid,          /* queue identifier */
    void *msg_buf,             /* message buffer */
    unsigned long msg_len,      /* length of message */
)
```

Description

This system call is identical in all respects to `q_vsend()` with one and only one exception: if one or more messages are already posted at the target queue, then the new message will be inserted into the queue's message queue in front of all such queued messages.

Arguments

<code>qid</code>	Specifies the queue ID of the target queue.
<code>msg_buf</code>	Points to the message to send.
<code>msg_len</code>	Specifies the length of the message. It must not exceed the queue's maximum message length.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Queue has been deleted.
0x06	ERR_OBJID	<code>qid</code> incorrect; failed validity check.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x30	ERR_KISIZE	Message buffer length exceeds max. KI packet buffer length.

Hex	Mnemonic	Description
0x31	ERR_MSGSIZ	Message is too large.
0x35	ERR_QFULL	Message queue at length limit.
0x3B	ERR_NOTVARQ	Queue is not variable length.
0x65	ERR_STALEID	Object's node has failed.

Notes

1. `q_vurgent()` is useful when the message represents an urgent errand and must be serviced ahead of the normally FIFO ordered messages.
2. If the caller is a task, it may be preempted as a result of this call.
3. The pSOS+ kernel must copy the message into a queue buffer or the receiving task's buffer. Longer messages take longer to copy. User's should account for the copy time in their design, especially when calling from an ISR.
4. `q_vurgent()` sends an urgent message to a variable length message queue. Use `q_urgent()` to send an urgent message to an ordinary queue.

Multiprocessor Considerations

1. If `qid` identifies a global queue residing on another processor node, the local kernel will internally make an RSC to that remote node to post the input message to that queue.
2. If a task awakened by this call does not reside on the local node, the local kernel will internally pass the message to the task's node of residence, whose pSOS+m kernel will ready the task and give it the relayed message. Thus, a `q_vurgent()` call, whether it is on the local or a remote queue, may cause pSOS+m activities on another processor node.

Callable From

- Task.
- ISR, if the target queue is local to the node from which the `q_vurgent()` call is made.
- KI, if the target queue is local to the node from which the `q_vurgent()` call is made.

- Callout, if the target queue is local to the node from which the `q_vurgent()` call is made.

See Also

`q_urgent`, `q_vreceive`, `q_vsend`

rn_create Creates a memory region.

```
#include <psos.h>
unsigned long rn_create(
    char name[4],           /* region name */
    void *saddr,            /* starting address */
    unsigned long length,    /* region's size in bytes */
    unsigned long unit_size, /* region's unit of allocation */
    unsigned long flags,     /* region attributes */
    unsigned long *rnid,     /* region ID */
    unsigned long *asiz      /* allocatable size */
)
```

1

Description

This service call enables a task to create a memory region, from which variable-sized memory segments can be allocated for use by the application. The pSOS+ kernel takes a portion from the beginning of this region to use as its Region Control Block (RNCB.) All relevant region arguments such as unit size and whether tasks will wait by FIFO or task priority order are established using this call.

Arguments

name	Specifies the user-assigned name of the new region.
saddr	Specifies the starting address of the region's memory area. saddr must be on a long word boundary.
length	Specifies the region's size, in bytes.
unit_size	Specifies the region's unit of allocation in bytes. unit_size must be a power of 2 and greater than or equal to 16. All allocation will be in multiples of unit_size.

flags	Specifies the region's attributes. <code>flags</code> is formed by OR-ing the following symbolic constants (one from each pair), which are defined in <code><psos.h></code> . For instance, to specify queuing by task priority, you place <code>RN_PRIOR</code> in <code>flags</code> . To specify queuing by task priority and to enable deletion of the region even if segments are allocated, you place both <code>RN_PRIOR</code> and <code>RN_DEL</code> in <code>flags</code> , using the following syntax: <div style="text-align: center;"><code>RN_PRIOR RN_DEL</code></div> <div style="text-align: center;"><code>RN_PRIOR / Tasks are queued by priority /FIFO order.</code></div> <div style="text-align: center;"><code>RN_FIFO</code></div> <div style="text-align: center;"><code>RN_DEL / Region can / cannot be deleted with segments</code></div> <div style="text-align: center;"><code>RN_NODEL outstanding.</code></div>
rnid	Points to the variable where <code>rn_create()</code> stores the region ID of the named region.
asiz	Points to the variable where <code>rn_create()</code> stores the actual number of allocatable bytes available in the region.

Return Value

This call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x08	ERR_OBJTFULL	Node's object table full.
0x1B	ERR_RNADDR	Starting address not on long word boundary.
0x1C	ERR_UNITSIZE	Illegal <code>unit_size</code> : not a power of 2 or not greater than or equal to 16.
0x1D	ERR_TINYUNIT	Length too large (for given <code>unit_size</code>).
0x1E	ERR_TINYRN	Cannot create; region length too small to hold RNCB.

Notes

1. Internally, the pSOS+ kernel treats a region name as a 32-bit integer. However, when the application calls the kernel through the pSOS+ C language API, it passes the region name as a four-byte character array.
2. The pSOS+ kernel does not check for duplicate region names. If duplicate names exist, an `rn_ident()` call can return the `rnid` of any region with the duplicate name.
3. A region must consist of physically contiguous memory locations.
4. The maximum input length for a region is 32767 times the region's unit size. A length that exceeds this limit for a given unit size is treated as an error, the remedy for which is a bigger unit size.
5. When the `RN_DEL` attribute is specified, a region can be deleted while segments are outstanding; otherwise, the pSOS+ kernel requires all segments to be returned before the region can be deleted.

Multiprocessor Considerations

Regions are strictly local resources, and cannot be exported. Therefore, any allocation calls must come only from the local node. However, if a region's memory is reachable from other nodes, then any segments allocated from it can be passed between nodes for direct access explicitly by the user's code.

Callable From

- Task

See Also

`rn_ident`, `rn_getseg`

rn_delete Deletes a memory region.

```
#include <psos.h>
unsigned long rn_delete (
    unsigned long rnid    /* region ID */
)
```

Description

This system call deletes the memory region with the specified region ID. Unless the region was created with the `RN_DEL` attribute set, `rn_delete()` is rejected if any segments allocated from the region have not been returned. The calling task does not have to be the creator of the region to be deleted.

Arguments

`rnid` Specifies the region ID of the region to be deleted.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Region already deleted.
0x06	ERR_OBJID	<code>rnid</code> is incorrect; failed validity check.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x1F	ERR_SEGINUSE	Cannot delete — one or more segments still in use.
0x27	ERR_TATRNDL	Informative only: tasks were waiting.

Notes

1. Once created, a region generally is used by multiple tasks for storing or passing data. Reasons for deleting a region are rare. Deleting a region is dangerous except as part of a partial or full system restart.
2. The special region with `rnid` equal to 0 cannot be deleted.

1

Multiprocessor Considerations

None, since regions are local resources. `rn_delete()` can be called only from the local node.

Callable From

- Task

See Also

`rn_create`

rn_getseg Allocates a memory segment to the calling task.

```
#include <psos.h>
unsigned long rn_getseg(
    unsigned long rnid,      /* region identifier */
    unsigned long size,      /* requested size, in bytes */
    unsigned long flags,     /* segment attributes */
    unsigned long timeout,   /* timeout in clock ticks */
    void **seg_addr         /* allocated segment address */
)
```

Description

This system call allocates a memory segment of the specified size from the specified memory region. An allocated segment's size is always the nearest multiple of the region's unit size, which is an input argument to the `rn_create()` call.

If the calling task selects the `RN_NOWAIT` attribute, then `rn_getseg()` returns unconditionally (whether or not allocation was successful). If the calling task elects the `RN_WAIT` attribute, and a subsequent request cannot be satisfied, the task is blocked until either a segment is allocated, or a timeout occurs (if the timeout attribute is elected).

Arguments

<code>rnid</code>	Specifies the region ID from which the memory segment is allocated.
<code>size</code>	Specifies the segment size, in bytes.
<code>flags</code>	Specifies the segment's attributes. The <code>flags</code> argument must assume one of the following values, defined in <code><psos.h></code> . <div style="margin-left: 20px;"> <code>RN_NOWAIT</code> Don't wait for a segment. <code>RN_WAIT</code> Wait for a segment. </div>
<code>timeout</code>	Specifies the timeout, in units of clock ticks. If <code>timeout</code> is 0 and <code>flags</code> is set to <code>RN_WAIT</code> , then <code>rn_getseg()</code> will wait forever. The <code>timeout</code> argument is ignored if <code>RN_NOWAIT</code> is used.
<code>seg_addr</code>	Points to the variable where <code>rn_getseg()</code> stores the starting address of the memory segment.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x01	ERR_TIMEOUT	Timed out; only if timeout requested.
0x05	ERR_OBJDEL	Region has been deleted.
0x06	ERR_OBJID	rnid is incorrect; failed validity check.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x20	ERR_ZERO	Cannot getseg; request size of zero is illegal.
0x21	ERR_TOOBIG	Request size too big for region.
0x22	ERR_NOSEG	No free segment; is returned only if RN_NOWAIT attribute is used.
0x26	ERR_RNKILLD	Region deleted while task waiting for segment.

Notes

1. See *pSOSystem System Concepts* for a description of the allocation algorithm for regions.
2. An allocated segment's size will always be a multiple of the region's unit size. It can, therefore, be greater than size.
3. An allocated segment always starts on a long word boundary.
4. If the calling task must wait, it will either wait by FIFO or priority order, depending on the attribute elected when the region was created.

Multiprocessor Considerations

Regions are strictly local resources, and cannot be exported. Therefore, any allocation calls must come only from the local node. However, if a region's memory

is reachable from other nodes, then any segments allocated from it can be passed between nodes for direct access explicitly by the user's code.

Callable From

- Task

See Also

`rn_create`, `rn_retseg`

rn_ident Obtains the region identifier of a named region.

```
#include <psos.h>
unsigned long rn_ident(
    char name[4],           /* region name */
    unsigned long *rnid     /* region identifier */
)
```

1

Description

This system call enables the calling task to obtain the region ID of a memory region for which the caller has only the region name. This region ID can then be used in all other operations relating to the memory region.

Most system calls, except `rn_create()` and `rn_ident()`, reference a region by its region ID. `rn_create()` returns the region ID to a region's creator. For other tasks, one way to obtain the region ID is to use `rn_ident()`.

Arguments

- `name` Specifies the user-assigned name of the region.
- `rnid` Points to the variable where `rn_ident()` stores the region ID of the named region.

Return Value

This call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x09	ERR_OBJNF	Named region not found.

Notes

1. Internally, the pSOS+ kernel treats a region name as a 32-bit integer. However, when the application calls the kernel through the pSOS+ C language API, it passes the region name as a four-byte character array.
2. The pSOS+ kernel does not check for duplicate region names. If duplicate names exist, an `rn_ident()` call can return the `rnid` of any region with the duplicate name.
3. The region with `rnid` equal 0 is special. This region is statically specified in the pSOS+ Configuration Table, and is used for pSOS+ data structures and task stacks.

Multiprocessor Considerations

None, since regions are strictly local resources. Only the local object table is searched.

Callable From

- Task

See Also

`rn_create`

rn_retseg Returns a memory segment to the region from which it was allocated.

```
#include <psos.h>
unsigned long rn_retseg(
    unsigned long rnid,    /* region identifier */
    void *seg_addr        /* segment address */
)
```

1

Description

This system call returns a memory segment back to the region from which it was allocated. The pSOS+ Region Manager then performs whatever compaction is possible, and puts the resulting free memory block in the region's free list for future allocation.

The segment address specified must be identical to the one returned by the original `rn_getseg()` call. Otherwise, the pSOS+ kernel will reject the segment.

Arguments

- `rnid` Specifies the segment's region of origin.
- `seg_addr` Specifies the starting address of the segment, as returned by `rn_getseg()`.

Return Value

This call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Region has been deleted.
0x06	ERR_OBJID	rnid is incorrect; failed validity check.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x23	ERR_NOTINRN	Segment does not belong to this region.

Hex	Mnemonic	Description
0x24	ERR_SEGADDR	Incorrect segment starting address.
0x25	ERR_SEGFREE	Segment is already unallocated.

Notes

1. Refer to *pSOSystem System Concepts* for the algorithms used to merge neighboring free segments.
2. There is no notion of segment ownership. A segment can be returned by a task other than the one that originally allocated it.
3. If there are tasks waiting for memory from this region, then such requests will be re-examined and allocation granted where possible — in the order of the wait queue (FIFO or by task priority).
4. Note that the calling task may be preempted if a task waiting for memory segment is unblocked as a result of the returned segment, and that task has higher priority.

Multiprocessor Considerations

None, since regions are strictly local resources. `rn_retseg()` can be called only from the local node.

Callable From

- Task

See Also

`rn_getseg`

sm_av (pSOS+m kernel only) Asynchronously releases a semaphore token.

```
#include <psos.h>
unsigned long sm_av(
    unsigned long smid    /* semaphore identifier */
)
```

1

Description

This system call is used to asynchronously release a semaphore token. It is identical to `sm_v()` except the call is made asynchronously. Refer to the description of `sm_v()` for further information. This call is only supported by the pSOS+m kernel (the multiprocessor version).

Arguments

`smid` Specifies the semaphore ID of the semaphore token to release.

Return Value

When called in a system running the pSOS+m kernel, this call always returns 0. The pSOS+ kernel (the single processor version) returns `ERR_SSFN`.

Error Codes

If the `sm_v()` call fails and the node's `MC_ASYNCERR` routine is present, that routine is invoked. The following error codes are possible:

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Semaphore has been deleted.
0x06	ERR_OBJID	<code>smid</code> incorrect; failed validity check.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.

If an `MC_ASYNCERR` routine is not provided, the pSOS+m kernel generates a fatal error.

Notes

The calling task can be preempted as a result of this call.

Multiprocessor Considerations

1. If `smid` identifies a global semaphore residing on another processor node, the pSOS+ kernel internally makes an RSC to that remote node to release the semaphore.
2. If the task awakened by this call does not reside on the local node, then the pSOS+m kernel internally alerts the task's node of residence, whose pSOS+m kernel will ready the task and give it the acquired semaphore token. Thus, an `sm_v()` call, whether it is to either the local or a remote semaphore, may cause pSOS+m activities on another processor node.

Callable From

- Task

See Also

`sm_v`, `sm_p`

sm_create Creates a semaphore.

```
#include <psos.h>
unsigned long sm_create(
    char name[4],           /* semaphore name */
    unsigned long count,    /* number of tokens */
    unsigned long flags,    /* semaphore attributes */
    unsigned long *smid     /* semaphore identifier */
)
```

1

Description

This system call creates a semaphore by allocating and initializing a Semaphore Control Block (SMCB) according to the specifications supplied with the call.

Like all objects, a semaphore has a user-assigned name, and a pSOS+-assigned semaphore ID returned by `sm_create()`. Several flag bits specify the characteristics of the semaphore, including whether tasks will wait for the semaphore by task priority or strictly FIFO.

Arguments

<code>name</code>	Specifies the user-assigned name of the new semaphore.
<code>count</code>	Specifies the initial semaphore token count.
<code>flags</code>	Specifies the semaphore's attributes. <code>flags</code> is formed by OR-ing the following symbolic constants (one from each pair), defined in <code><psos.h></code> : <div style="margin-left: 20px;"> <code>SM_GLOBAL</code> / Semaphore can be addressed by other nodes / <code>SM_LOCAL</code> local nodes only. <code>SM_PRIOR</code> / Tasks are queued by priority FIFO order. <code>SM_PRIOR</code> </div>
<code>smid</code>	Points to the variable where <code>sm_create()</code> stores the semaphore ID of the named semaphore.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x08	ERR_OBJTFULL	Node's object table is full.
0x41	ERR_NOSCB	Exceeds node's maximum number of semaphores.

Notes

1. Internally, the pSOS+ kernel treats a semaphore name as a 32-bit integer. However, when the application calls the kernel through the pSOS+ C language API, it passes the semaphore name as a four-byte character array.
2. The pSOS+ kernel does not check for duplicate semaphore names. If duplicate names exist, an `sm_ident()` call can return the `smid` of any semaphore with the duplicate name.
3. The maximum number of semaphores that can be simultaneously active is defined by the `kc_nsema4` entry in the pSOS+ Configuration Table.
4. The `count` argument is unsigned, and thus can only be 0 or a positive value.
5. The `SM_GLOBAL` attribute is ignored by the single-processor version of the pSOS+ kernel.

Multiprocessor Considerations

1. The `SM_GLOBAL` attribute should be set only if the semaphore must be made known to other processor nodes in a multiprocessor configuration. If set, the semaphore's name and `smid` are sent to the master node for entry in its Global Object Table.
2. If the `SM_GLOBAL` attribute is set and the number of global objects currently exported by the node equals the Multiprocessor Configuration Table entry `mc_nglbobj` then the semaphore is not created and `ERR_OBJTFULL` is returned.

Callable From

- Task

See Also

`sm_delete`, `sm_ident`

sm_delete

Deletes a semaphore.

```
#include <psos.h>
unsigned long sm_delete(
    unsigned long smid    /* semaphore ID */
)
```

Description

This system call deletes the semaphore with the specified semaphore ID, and frees the SMCB to be reused. `sm_delete()` takes care of cleaning up the semaphore. If there are tasks waiting, they will be unblocked and given an error code.

The calling task does not have to be the creator (parent) of the semaphore to be deleted. However, a semaphore must be deleted from the node on which it was created.

Arguments

`smid` Specifies the semaphore ID of the semaphore to be deleted.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Semaphore has been deleted.
0x06	ERR_OBJID	<code>smid</code> incorrect; failed validity check.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x44	ERR_TATSDEL	Informative only; there were tasks waiting.
0x53	ERR_ILLRSC	Semaphore not created from this node.

Notes

1. Once created, a semaphore is generally used by multiple tasks for communication and synchronization. There is rarely a reason for deleting a semaphore, even when it is no longer used, except to allow reuse of the SMCB.
2. The calling task can be preempted, if a task waiting at the deleted semaphore has higher priority.

1

Multiprocessor Considerations

If `smid` identifies a global semaphore, `sm_delete` will notify the master node so that the semaphore can be removed from its Global Object Table. Thus, deletion of a global semaphore always causes activity on the master node.

Callable From

- Task

See Also

`sm_create`

sm_ident Obtains the semaphore identifier of a named semaphore.

```
#include <psos.h>
unsigned long sm_ident(
    char name[4],          /* semaphore name */
    unsigned long node,    /* node selector */
    unsigned long *smid    /* semaphore ID */
)
```

Description

This system call enables the calling task to obtain the semaphore ID of a semaphore it only knows by name. The semaphore ID can then be used in all other operations relating to this semaphore.

Most system calls, except `sm_create()` and `sm_ident()`, reference a semaphore by the semaphore ID. `sm_create()` returns the semaphore ID to the semaphore creator. For other tasks, one way to obtain the semaphore ID is to use `sm_ident()`.

Arguments

name	Specifies the user-assigned name of the semaphore.
node	In multiprocessor systems, is a search order specifier. See “Multiprocessor Considerations.” In a single node system, this argument must be 0.
smid	Points to the variable where <code>sm_ident()</code> stores the semaphore ID of the named semaphore.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x04	ERR_NODENO	Node specifier out of range.
0x09	ERR_OBJNF	Named semaphore not found.

Notes

1. Internally, the pSOS+ kernel treats a semaphore name as a 32-bit integer. However, when the application calls the kernel through the pSOS+ C language API, it passes the semaphore name as a four-byte character array.
2. The pSOS+ kernel does not check for duplicate semaphore names. If duplicate semaphore names exist, an `sm_ident()` call can return the `smid` of any semaphore with the duplicate name.

Multiprocessor Considerations

1. `sm_ident()` converts a semaphore's name to its `smid` by using a search order determined by the `node` input parameter, as described in *pSOSystem System Concepts*. Because semaphores created and exported by different nodes may not have unique names, the result of this binding may depend on the order in which the object tables are searched.
2. If the master node's Global Object Table must be searched, then the pSOS+m kernel makes a `sm_ident()` RSC to the master node.

Callable From

- Task

See Also

`sm_create`

sm_p Acquires a semaphore token.

```
#include <psos.h>
unsigned long sm_p(
    unsigned long smid,    /* semaphore identifier */
    unsigned long flags,   /* attributes */
    unsigned long timeout /* timeout */
)
```

Description

This system call enables a task or an ISR to acquire a semaphore token. A calling task can specify whether or not it wants to wait for the token. If the semaphore token count is positive, then this call returns the semaphore token immediately. If the semaphore token count is zero and the calling task specified `SM_NOWAIT`, then `sm_p()` returns with an error code. If `SM_WAIT` is elected, the task will be blocked until a semaphore token is released, or if the `timeout` argument is specified, until timeout occurs, whichever occurs first.

Arguments

smid	Specifies the semaphore ID of the semaphore token.				
flags	Specifies whether <code>sm_p()</code> will block waiting for a token. <code>flags</code> should have one of the following values (defined in <code><psos.h></code>): <table data-bbox="399 1067 1202 1171"> <tr> <td><code>SM_WAIT</code></td><td>Block until semaphore is available.</td></tr> <tr> <td><code>SM_NOWAIT</code></td><td>Return with error code if semaphore is unavailable.</td></tr> </table>	<code>SM_WAIT</code>	Block until semaphore is available.	<code>SM_NOWAIT</code>	Return with error code if semaphore is unavailable.
<code>SM_WAIT</code>	Block until semaphore is available.				
<code>SM_NOWAIT</code>	Return with error code if semaphore is unavailable.				
timeout	Specifies the timeout interval, in units of clock ticks. If <code>timeout</code> is zero and <code>flags</code> is set to <code>SM_WAIT</code> , then <code>sm_p()</code> will wait forever. <code>timeout</code> will be ignored if <code>flags</code> is set to <code>SM_NOWAIT</code> .				

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x01	ERR_TIMEOUT	Timed out; this error code is returned only if a timeout was requested.
0x05	ERR_OBJDEL	Semaphore has been deleted.
0x06	ERR_OBJID	smid incorrect; failed validity check.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x42	ERR_NOSEM	No semaphore; this error code is returned only if SM_NOWAIT was selected.
0x43	ERR_SKILLD	Semaphore deleted while task waiting.
0x54	ER_NOAGNT	Cannot wait on remote object; no free agents at node.
0x65	ERR_STALEID	Object's node has failed.
0x66	ERR_NDKLD	Object's node failed while RSC waiting.

Notes

If it is necessary to block the calling task, `sm_p()` will enter the calling task in the semaphore's task-wait queue. If the semaphore was created with the `SM_FIFO` attribute, then the task is simply entered at the tail of the wait queue. If the semaphore was created with the `SM_PRIOR` attribute, then the task is inserted into the wait queue by priority.

Multiprocessor Considerations

If `smid` identifies a global semaphore residing on another processor node, the local kernel will internally make an RSC to that remote node to acquire a semaphore token. If the `SM_WAIT` attribute is used, then the pSOS+ kernel on the target node must use an agent to wait for the semaphore token. If that node is temporarily out of agents, the call will fail. The number of agents on each node is defined by the `mc_nagent` entry in the node's Multiprocessor Configuration Table.

Callable From

- Task.
- ISR, if SM_NOWAIT is set and the semaphore is local to the node from which the `sm_p()` call is made.
- KI, if SM_NOWAIT is set and the semaphore is local to the node from which the `sm_p()` call is made.
- Callout, if SM_NOWAIT is set and the semaphore is local to the node from which the `sm_p()` call is made.

See Also

`sm_v`

sm_v Releases a semaphore token.

```
#include <psos.h>
unsigned long sm_v(
    unsigned long smid     /* semaphore identifier */
)
```

1

Description

This system call is used to release a semaphore token. If a task is already waiting at the semaphore, it is unblocked and made ready to run. If there is no task waiting, then the semaphore token count is simply incremented by 1.

Arguments

smid Specifies the semaphore ID of the semaphore token to release.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Semaphore has been deleted.
0x06	ERR_OBJID	smid incorrect; failed validity check.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x65	ERR_STALEID	Object's node has failed.

Notes

If the caller is a task, it may be preempted as a result of this call.

Multiprocessor Considerations

1. If `smid` identifies a global semaphore residing on another processor node, then the pSOS+ kernel will internally make an RSC to that remote node to release the semaphore.
2. If the task awakened by this call does not reside on the local node, the local kernel will internally alert the task's node of residence, whose pSOS+ kernel will ready the task and give it the acquired semaphore token. Thus, an `sm_v()` call, whether it is to a local or remote semaphore, may cause pSOS+ activities on another node.

Callable From

- Task.
- ISR, if semaphore is local to the node from which the `sm_v()` call is made.
- KI, if the semaphore is local to the node from which the `sm_v()` call is made.
- Callout, if the semaphore is local to the node from which the `sm_v()` call is made.

See Also

`sm_p`

t_create Creates a task.

```
#include <psos.h>
unsigned long t_create(
    char name[4],           /* task name */
    unsigned long prio,     /* task priority */
    unsigned long sstack,   /* task supervisor stack size */
    unsigned long ystack,   /* task user stack size */
    unsigned long flags,    /* task attributes */
    unsigned long *tid      /* task identifier */
)
```

1

Description

This service call enables a task to create a new task. `t_create()` allocates to the new task a Task Control Block (TCB) and a memory segment for its stack(s). The task stack sizes and scheduling priority are established with this call. `t_create()` leaves the new task in a dormant state; the `t_start()` call must be used to place the task into the ready state for execution.

Arguments

name	Specifies the user-assigned name of the task.
prio	Specifies the task's initial priority within the range 1 - 239, with 239 the highest and 1 the lowest. Priority level 0 is reserved for the pSOS+ daemon task IDLE. Priority levels 240 - 255 are reserved for a variety of high priority pSOSsystem daemon tasks. While <code>t_create()</code> will allow creation of a task at these priorities, there should never be a need to do so.
sstack	Specifies the task's supervisor stack size in bytes (see "Supervisor Stack Size" under "Target.") <code>t_create()</code> internally calls <code>rn_getseg()</code> to allocate a segment from Region 0 to hold the task's stack and the user stack, if any.
ystack	Specifies the task's user stack. <code>ystack</code> may be 0 if the task executes only in supervisor mode (see "Using sstack and ystack" under "Target").

flags	Specifies the task's attributes. <code>flags</code> is formed by OR-ing the following symbolic constants (one from each pair), which are defined in <code><psos.h></code> . For instance, to specify that a task is global, you place the symbolic constant <code>T_GLOBAL</code> in <code>flags</code> . To specify that the task is global and uses the FPU processor, you place both <code>T_GLOBAL</code> and <code>T_FPU</code> in <code>flags</code> , using the following syntax: <div style="text-align: center;"><code>T_GLOBAL T_FPU</code></div> <div style="display: flex; justify-content: space-between;"><div><code>T_GLOBAL /</code> <code>T_LOCAL</code></div><div>Makes the task global: external tasks on other nodes can address it / restricts the task to the local node. The <code>T_GLOBAL</code> attribute is ignored by the single-processor kernel.</div></div> <div style="display: flex; justify-content: space-between;"><div><code>T_FPU /</code> <code>T_NOFPU</code></div><div>Informs the pSOS+ kernel that the task uses /does not use the FPU coprocessor (see "Using the <code>T_FPU</code> Flag" under "Target.")</div></div>
tid	Points to the variable where <code>t_create()</code> stores the task ID assigned to the task.

Target

Using `sstack` and `ustack`

On most processors, a task can execute only in supervisor mode. Thus, a task can have only a supervisor stack. On these processors `ustack` is added to `sstack` to create a supervisor stack of the combined sizes. Exceptions to this usage are shown below.

68K	On 68K processors, a task can execute in either user mode or supervisor mode. A user stack is not needed if the task never executes in the user mode, in which case <code>ustack</code> should be set to 0. If the task starts in the user mode, then <code>ustack</code> must be greater than 20 bytes. The supervisor/user mode is set in the <code>t_start()</code> system call.
CF	On ColdFire and PowerPC processors, a task can execute in either user mode or supervisor mode, but there are not separately defined stacks depending on this mode. <code>t_create()</code> simply adds <code>sstack</code> and <code>ustack</code> together and allocates a stack of that resultant size.
PPC	



On ARM processors, a task can execute in either user mode or supervisor mode. A user stack is not needed if the task never executes in the user mode, in which case `ustack` is set to 0. If the task starts in the user mode, then `ustack` must be greater than 80 bytes to accommodate the requirements of the interrupt handler. The supervisor/user mode is set in the `t_start()` system call.

Supervisor Stack Size

Supervisor stack size is processor-dependent.



On PowerPC and ARM processors, the stack size should be no less than 512 bytes.



On 960 processors, the stack size should be no less than 256 bytes.



On x86 processors, the stack size should be no less than 128 bytes.

Using the T_FPU Flag

If the `T_FPU` flag is set, the size of the task's stack segment is extended to save and restore FPU registers. It should be set if the task uses the FPU. The extension added varies according to the processor being used.



On 68K processors, the stack is extended by 328 bytes.



On ColdFire processors, there is no support for FPU. Hence this feature is not supported in pSOS+.



On PowerPC processors, the stack is extended by 264 bytes.



On ARM processors, the stack is extended by 100 bytes.



On 960 processors, the stack is extended by 48 bytes.



On 486 processors, and on 386 processors used with an 80387 FPU, the stack size is extended by 108 bytes.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x08	ERR_OBJTFULL	Node's object table full.
0x0E	ERR_NOTCB	Exceeds node's maximum number of tasks. The maximum number of tasks that can be simultaneously active is defined by the <code>kc_ntask</code> entry in the pSOS+ Configuration Table.
0x0F	ERR_NOSTK	Insufficient space in Region 0 to create stack.
0x10	ERR_TINYSTK	Stack size too small.
0x11	ERR_PRIOR	Priority out of range.

Notes

1. Internally, the pSOS+ kernel treats a task name as a 32-bit integer. However, when the application calls the kernel through the pSOS+ C language API, it passes the task name as a four-byte character array.
2. A null name (for example, 32-bit binary 0) should not be used, because it may be used elsewhere as an alias for the running task.
3. The pSOS+ kernel does not check for duplicate task names. If duplicate names exist, a `t_ident()` call can return the `tid` of any task with the duplicate name.

4. If you have installed any other components from Integrated Systems, the pSOS+ kernel adds an extension for each component to the task's stack segment. These extension sizes can be determined from the user manuals for those components.

Multiprocessor Considerations

1. The `T_GLOBAL` attribute should be set only if the task must be made known to other processor nodes in a multiprocessor configuration. If set, the task's name and `tid` are sent to the master node for entry in its Global Object Table.
2. If the `T_GLOBAL` attribute is set and the number of global objects currently exported by the node equals the Multiprocessor Configuration Table entry `mc_nglbobj`, then the task is not created and `ERR_OBJTFULL` is returned.

Callable From

- Task

See Also

`t_start`, `t_ident`, `rn_getseg`

t_delete Deletes a task.

```
#include <psos.h>
unsigned long t_delete(
    unsigned long tid    /* task identifier */
)
```

Description

This service call enables a task to delete itself or another task. The pSOS+ kernel halts the task and reclaims its TCB, stack segment and any allocated timers.

The calling task does not have to be the creator (parent) of the task to be deleted. However, a task must be deleted from the node on which it was created.

Arguments

`tid` Specifies the task ID of the task to be deleted.

Return Value

This call returns 0 on success (unless the caller does a self-delete, in which case the call does not return) or returns an error on failure.

Error Codes

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Task already deleted.
0x06	ERR_OBJID	<code>tid</code> incorrect; failed validity check.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x18	ERR_DELF5	pHILE+ resources in use.
0x19	ERR_DELLC	pREPC+ resources in use.
0x1A	ERR_DELNS	pNA+ resources in use.
0x53	ERR_ILLRSC	Task not created from this node.

Notes

1. If the call is to delete self (suicide via `tid` equal to 0), there will be no return.
2. Task deletion should be carefully planned and considered. Indiscriminate use can lead to unpredictable results, especially when resources such as allocated memory segments, buffers, or semaphores have not been correctly returned. If a task holds any resources from the pREPC+ library, the pHILE+ file system manager, or the pNA+ network manager, those resources must be returned before `t_delete()` is called. The commands that must be executed for pREPC+, pHILE+, and pNA+ resources are `fclose(0)`, `close_f(0)`, and `close(0)`, respectively. Following these commands, a `free()` command must be used to return pREPC+ memory. This order of execution is required because the pREPC+ library calls the pHILE+ file system manager, and the pHILE+ file system manager calls the pNA+ network manager (if NFS is in use.) If resources are not returned in the correct order, an error occurs. See Error Codes.

The following is an example of the correct sequence of calls for returning resources. This example applies to a case where all three components have allocated resources:

```
fclose(0); /* return pREPC+ resources */
close_f(0); /* return pHILE+ resources */
close(0); /* return pNA+ resources */
free(-1) /* return pREPC+ memory */
t_delete(0); /* and execute self-deletion */
```

3. Using `t_delete()` to delete another task does not generally allow that task a chance to perform any cleanup work necessary for the orderly termination of the task. In general, `t_delete()` should be used for self-deletion, whereas asynchronous deletion of another task can best be achieved using the `t_restart()` call.
4. `t_delete()` calls the optional user-supplied callout procedure, whose address is defined in the `kc_deleteco` entry in the pSOS+ Configuration Table.

Multiprocessor Considerations

1. A task can be deleted only from the local node.
2. If `tid` identifies a global task, `t_delete` notifies the master node so that the task can be removed from its Global Object Table. Thus, deletion of a global task always causes activity on the master node.

Callable From

- Task

See Also

t_restart

t_getreg Gets a task's notepad register.

```
#include <psos.h>
unsigned long t_getreg(
    unsigned long tid,          /* task identifier */
    unsigned long regnum,      /* register number */
    unsigned long *reg_value   /* register contents */
)
```

1

Description

This system call enables the caller to obtain the contents of a task's notepad register. Each task has 16 such software registers, held in the task's TCB. The purpose of these registers is to furnish every task with a set of named, permanent variables. Eight of these registers are reserved for system use. Eight are free to be used for application specific purposes.

Arguments

tid	Specifies the task ID of the task whose notepad register will be read. If tid equals 0, then the calling task reads its own notepad register.
regnum	Specifies the register number. Registers numbered 0 through 7 are for application use, and registers 8 through 15 are reserved for system purposes.
reg_value	Points to the variable where t_getreg() stores the register's contents.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Task has been deleted.
0x06	ERR_OBJID	tid is incorrect; failed validity check.

Hex	Mnemonic	Description
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x17	ERR_REGNUM	Register number is out of range.
0x65	ERR_STALEID	Object's node has failed.

Notes

The kernel does not deny access to those registers reserved for system use. For future compatibility, however, you should not use them.

Multiprocessor Considerations

If the `tid` identifies a global task residing on another processor node, the local kernel will internally make an RSC to that remote node to obtain the register's content for that task.

Callable From

- Task.
- ISR, if the task is local to the node from which the `t_getreg()` call is made.
- KI, if the task is local to the node from which the `t_getreg()` call is made.
- Callout, if the task is local to the node from which the `t_getreg()` call is made.

See Also

`t_setreg`

t_ident Obtains the task identifier of a named task.

```
#include <psos.h>
unsigned long t_ident(
    char name[4],           /* task name */
    unsigned long node,     /* node number */
    unsigned long *tid      /* task ID */
)
```

1

Description

This system call enables the calling task to obtain the task ID of a task it knows only by name. This task ID can then be used in all other operations relating to the task.

Most system calls, except `t_create()` and `t_ident()`, reference a task by its task ID. `t_create()` returns the task ID to a task's creator. For other tasks, one way to obtain the task ID is to use `t_ident()`.

Arguments

name	Specifies the user-assigned name of the task.
node	In multiprocessor systems, is a search order specifier. See "Multiprocessor Considerations." In a single node system, this argument must be 0.
tid	Points to the variable where <code>t_ident()</code> stores the task ID of the named task.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x04	ERR_NODENO	Illegal node number.
0x09	ERR_OBJNF	Named task was not found.

Notes

1. Internally, the pSOS+ kernel treats a task name as a 32-bit integer. However, when the application calls the kernel through the pSOS+ C language API, it passes the task name as a four-byte character array.
2. The pSOS+ kernel does not check for duplicate task names. If duplicate task names exist, a `t_ident` call can return the `tid` of any task with the duplicate name.
3. If `name` is null (for example, `(char*)0`), then the `tid` of the calling task is returned.

Multiprocessor Considerations

1. `t_ident()` converts a task's name to its `tid` using a search order determined by the `node` input parameter, as described in *pSOSSystem System Concepts*. Because tasks created and exported by different nodes may not have unique names, the result of this binding may depend on the order in which the object tables are searched.
2. If the master node's Global Object Table must be searched, then the pSOS+m kernel makes a `t_ident()` RSC to the master node.
3. If the task name is null (i.e., `(char*)0`), then the `node` argument is ignored, and the `t_ident()` operation returns the `tid` of the calling task on the local node.

Callable From

- Task

See Also

`t_create`

t_mode Gets or changes the calling task's execution mode.

```
#include <psos.h>
unsigned long t_mode(
    unsigned long mask,      /* attributes to be changed */
    unsigned long new_mode,  /* new attributes */
    unsigned long *old_mode /* prior mode */
)
```

1

Description

This system call enables a task to modify certain execution mode fields. These are preemption on/off, roundrobin on/off, asynchronous signal handling on/off, and interrupt control.

Preemption has precedence over timeslicing. Therefore, if preemption is off, timeslicing does not occur whether or not it is set.

The calling task can be preempted as a result of this call, if its preemptibility is turned from off to on and a higher priority task is ready to run.

To obtain a task's current execution mode without changing it, use a `mask` of 0.

Arguments

<code>mask</code>	Specifies all task attributes to be modified.
<code>new_mode</code>	Specifies the new task attributes.
<code>old_mode</code>	Points to the variable where <code>t_mode()</code> stores the old value of the task's mode.

You create the arguments `mask` and `new_mode` by ORing symbolic constants from the pairs below. These symbolic constants are also defined in `<psos.h>`.

<code>T_PREEMPT / T_NOPREEMPT</code>	Task is / is not preemptible.
<code>T_TSLICE / T_NOTSLICE</code>	Task can / cannot be time-sliced.
<code>T_ASR / T_NOASR</code>	Task's ASR is enabled / disabled.

T_ISR / T_NOISR	Hardware interrupts are enabled / disabled while the task runs. These options are available only on certain processors. See “Interrupt Control” under “Target.”
T_LEVELMASK0 through T_LEVELMASK _n	Certain hardware interrupts are disabled while the task runs. These options are available only on certain processors. See “Interrupt Control” under “Target.”

To create the argument `new_mode`, you pick symbolic constants from the pairs described above. For instance, to specify that a task have preemption turned off, you place the symbolic constant `T_NOPREEMPT` in `new_mode`. To specify that the task have preemption turned off and roundrobin by time-slicing turned on, you place both `T_NOPREEMPT` and `T_TSLICE` in `new_mode`, using the following syntax:

```
T_NOPREEMPT | T_TSLICE
```

The argument `mask` specifies the bit mask used to permit attribute modifications, and as such, it must contain *both* of the symbolic constants from each pair whose attribute is to be modified. For instance, to enable modification of preemption mode, you place both `T_PREEMPT` and `T_NOPREEMPT` in `mask`. To enable modification of both preemption mode and roundrobin mode, you place both symbolic constants from both pairs in `mask`, as below:

```
T_PREEMPT | T_NOPREEMPT | T_NOTSLICE | T_TSLICE
```

Target

Interrupt Control

Interrupt control means that while a task is executing, hardware interrupts are disabled. On some processors, you can disable all interrupts at or below a certain interrupt level and enable all interrupts above that level. On other processors you

can simply specify that all interrupts are either enabled or disabled. Details are provided below:

On PowerPC, x86, MIPS, and ARM processors, you can simply enable or disable all hardware interrupts. To do this, you place both `T_ISR` and `T_NOISR` in the `mask` argument and place either `T_ISR` or `T_NOISR` in the `new_mode` argument.



NOTE: `t_mode()` stores in `old_mode` the previous setting of the interrupt control value as stored in the task's TCB (called the true mode), rather than the value in the task's processor status register (called the *transient* mode.) These two modes are normally the same. The one instance when they can be different is if the task changes the interrupt control value without using `t_mode()`.

Processor Mode

`t_mode()` cannot modify the task's processor mode. Most processors only have one processor mode, so this is not relevant. Exceptions are handled as follows:

Return Value

This system call always returns 0.

Error Codes

None.

Notes

Multiprocessor Considerations

None. Because `t_mode()` affects only the calling task, its action stays on the local node.

Callable From

- ## ■ Task

See Also

t_start

t_restart Forces a task to start over regardless of its current state.

```
#include <psos.h>
unsigned long t_restart(
    unsigned long tid,          /* task identifier */
    unsigned long targs[4]     /* startup arguments */
)
```

1

Description

This system call forces a task to resume execution at its original start address regardless of its current state or place of execution. If the task was blocked, the pSOS+ kernel forcibly unblocks it. The task's priority and stacks are set to the original values that `t_create()` specified. Its start address and execution mode are reset to the original values established by `t_start()`. Any pending events, signals, or armed timers are cleared.

The `t_restart()` call accepts a new set of up to four arguments, which, among other things, can be used by the task to distinguish between the initial startup and subsequent restarts.

Because it can unconditionally unblock a task and alter its flow of execution, `t_restart()` is useful for forcing a task to execute cleanup code on its own behalf after which the task can delete itself by executing `t_delete()`.

The calling task does not have to be the creator (or parent) of the task it restarts. However, a task must be restarted from the node on which it was created.


Arguments


<code>tid</code>	Specifies the task to restart. When <code>tid</code> equals 0, the calling task restarts itself.
<code>targs</code>	Specifies up to four long words of input that are passed to the restarted task.

Target


Startup Values

At the start of the task, the CPU registers and the stack are initialized in such a way that if the outermost function of the task exits by mistake, an illegal error address results. The contents of the original registers and stack are platform-specific:

 On 960 processors, the return address (RIP) on the stack is 0xDEADDEAD.

 On PowerPC and ARM processors, the return address in register LR is 0xDEADDEAD.



 On x86 processors, the return address on the stack is 0xFFFFFFFF.

A restarted task can receive up to four long words of input arguments. To facilitate retrieval of these arguments, they are passed to the task as if it is invoked as a high-level language procedure or function. For example, if a C task `nice` has three input arguments, it can be declared as follows:

```
nice (unsigned long a, unsigned long b, unsigned long c);
```

where `targs[0]` is passed to `a`, `targs[1]` to `b`, and `targs[2]` to `c`. In this case, `targs[3]` is irrelevant and does not need the calling task to load it.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Task already deleted.
0x06	ERR_OBJID	tid is incorrect: failed validity check.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x0D	ERR_RSTFS	Information only: possible file corruption on restart.
0x13	ERR_NACTIVE	Cannot restart: this task never started.
0x53	ERR_ILLRSC	Task not created on this node.

1

Notes

1. Even though `t_restart()` resets the task's stacks, the stacks' contents remain intact. The stack frame and any automatic variables it contains for the task's outermost procedure should remain intact (despite restart) until something is stored into it, or it is initialized.
2. Once restarted, a task can actively compete for the CPU and all other system resources. Thus, it can preempt the calling task if it has a higher priority.

Multiprocessor Considerations

None. A task can be restarted from the local node only.

Callable From

- Task

See Also

`t_create`, `t_start`, `t_delete`

t_resume Resumes a suspended task.

```
#include <psos.h>
unsigned long t_resume(
    unsigned long tid    /* task identifier */
)
```

Description

This system call removes the suspension of a task. If the task was suspended while in the ready state, `t_resume()` releases it to be scheduled for execution. If the task was both suspended and blocked (for example, waiting for a message), `t_resume()` removes only the suspension. This leaves the task in the blocked state.

Arguments

`tid` Specifies the task ID of the task.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Task already deleted.
0x06	ERR_OBJID	tid incorrect: failed validity check.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x15	ERR_NOTSUSP	Task not suspended.
0x65	ERR_STALEID	Object's node has failed.

Notes

The calling task can be preempted as a result of this call.

Multiprocessor Considerations

If `tid` identifies a global task residing on another processor node, the local kernel internally makes an RSC to that remote node to resume the task.

Callable From

- Task.
- ISR, if the task is local to the node from which the `t_resume()` call is made.
- KI, if the task is local to the node from which the `t_resume()` call is made.
- Callout, if the task is local to the node from which the `t_resume()` call is made.

See Also

`t_suspend`

t_setpri

Gets and optionally changes a task's priority.

```
#include <psos.h>
unsigned long t_setpri(
    unsigned long tid,      /* task identifier */
    unsigned long newprio,  /* new priority */
    unsigned long *oldprio /* previous priority */
)
```

Description

This system call enables the calling task to obtain and optionally modify either its own or another task's scheduling (software) priority. The previous priority is returned.

Arguments

tid	Specifies the selected task for the priority change. If tid equals 0, the calling task is the target.
newprio	Specifies the task's new priority. newprio must be between 0 and 255 (see Note 3). If newprio is 0, the task's priority is not changed. This allows a read of a task's priority without changing its priority.
oldprio	Points to the variable where t_setpri() stores the task's previous priority.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	The task was already deleted.
0x06	ERR_OBJID	tid is incorrect: failed validity check.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.

Hex	Mnemonic	Description
0x16	ERR_SETPRI	Cannot change: new priority value is out of range.
0x65	ERR_STALEID	Object's node has failed.

Notes

1. If the calling task uses `t_setpri()` to lower its own priority, it can be preempted by a ready task with higher priority.
2. If the calling task uses `t_setpri()` to raise the priority of another task, it can be preempted if that task is ready and now possesses higher priority.
3. Priority level 0 is reserved for the pSOS+ daemon task IDLE. Priority levels 240 - 255 are reserved for a variety of high priority pSOSystem daemon tasks. While `t_create()` will allow creation of tasks at these priorities, there should never be a need to do so.

Multiprocessor Considerations

If the `tid` identifies a global task residing on another processor node, the local kernel internally makes an RSC to that remote node to change the priority of the task.

Callable From

- Task

See Also

`t_create`

t_setreg

Sets a task's notepad register.

```
#include <psos.h>
unsigned long t_setreg(
    unsigned long tid,      /* task identifier */
    unsigned long regnum,   /* register number */
    unsigned long reg_value /* register value */
)
```

Description

This system call enables the caller to modify the contents of a task's notepad register. Each task has 16 such software registers, held in the task's TCB. The purpose of these registers is to furnish every task with a set of named, permanent variables. Eight of these registers are reserved for system use, and eight are free to be used for application-specific purposes.

Arguments

tid	Specifies the task ID of the task whose notepad register is set. If tid equals 0, the calling task sets its own notepad register.
regnum	Specifies the register number. Registers 0 through 7 are for application use, and registers 8 through 15 are reserved for system use.
reg_value	Specifies the value at which the register's contents are to be set.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Task already deleted.
0x06	ERR_OBJID	tid incorrect: failed validity check.

Hex	Mnemonic	Description
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x17	ERR_REGNUM	Register number out of range.
0x65	ERR_STALEID	Object's node has failed.

Notes

The kernel does not deny access to the registers that the system reserves. For future compatibility, however, avoid using these reserved registers.

Multiprocessor Considerations

If `tid` identifies a global task residing on another processor node, the local kernel internally makes an RSC to that remote node to set the register for the task.

Callable From

- Task.
- ISR, if the task is local to the node from which the `t_setreg()` call is made.
- KI, if the task is local to the node from which the `t_setreg()` call is made.
- Callout, if the task is local to the node from which the `t_setreg()` call is made.

See Also

`t_getreg`

t_start Starts a task.

```
#include <psos.h>
unsigned long t_start(
    unsigned long tid,    /* task identifier */
    unsigned long mode,   /* initial task attributes */
    void (*start_addr)(), /* task address */
    unsigned long targs[4] /* startup task arguments */
)
```

Description

This system call places a newly created task into the ready state to await scheduling for execution. The calling task does not have to be the creator (or parent) of the task to be started. However, a task must be started from the node on which it was created.

Arguments

tid	Specifies the task to start. tid is returned by the t_create() and t_ident() calls.
mode	Specifies the initial task attributes. mode is formed by ORing the following symbolic constants (one from each pair), which are defined in <psos.h>. For instance, to specify that a task should have preemption turned off, you place the symbolic constant T_NOPREEMPT in mode. To specify that the task should have preemption turned off and roundrobin by time-slicing turned on, you place both T_NOPREEMPT and T_TSLICE in mode, using the following syntax: <div style="margin-left: 40px;"> T_NOPREEMPT T_TSLICE </div> <div style="margin-left: 40px;"> T_PREEMPT / Task is / is not preemptible. T_NOPREEMPT </div> <div style="margin-left: 40px;"> T_TSLICE / Task can /cannot be time-sliced. T_NOTSLICE </div> <div style="margin-left: 40px;"> T_ASR / Task's ASR is enabled / disabled. T_NOASR </div>

T_USER / T_SUPV	Task runs in user / supervisor mode. (See “User and Supervisor Modes” under “Target.”)
T_ISR / T_NOISR	Hardware interrupts are enabled / disabled while task runs. These options are available only on certain processors. See “Interrupt Control” under “Target.”
T_LEVELMASK0 through T_LEVELMASKn	Certain hardware interrupts are disabled while the task runs. These options are available only on certain processors. See “Interrupt Control” under “Target”.
start_addr	Specifies the task's location in memory.
targs	Specifies four startup values passed to the task (see “Startup Values” under “Target”).

Target

Startup Values

At the start of the task, the CPU registers and the stack are initialized in such a way that if the outermost function of the task exits by mistake, an illegal error address results. The contents of the original registers and stack are platform-specific:

- 960

On 960 processors, the return address (RIP) on the stack is 0xDEADDEAD.
- PPC

On PowerPC and ARM processors, the return address in register LR is 0xDEADDEAD.
- ARM
- x86

On x86 processors, the return address on the stack is 0xFFFFFFFF.

A new task can receive up to four long words of input arguments. To facilitate retrieval of these arguments, they are passed to the task as if it is invoked as a high-level language procedure or function. For example, if a C task `nice` has three input arguments, it can be declared as follows:

```
nice (unsigned long a, unsigned long b, unsigned long c);
```

where `targs[0]` is passed to `a`, `targs[1]` to `b`, and `targs[2]` to `c`. In this case, `targs[3]` is irrelevant and does not need the calling task to load it.

User and Supervisor Modes

You use the symbolic constants `T_USER` and `T_SUPV` on each processor as follows:

On 960 and x86 processors, a task can execute only in supervisor mode. In `<psos.h>`, for these processors, the symbols `T_SUPV` and `T_USER` are defined as:

960	#define T_SUPV 0
x86	#define T_USER 0

for the sole purpose of compatibility with platforms that support both user and supervisor modes.

Interrupt Control

Interrupt control means that while a task is executing, hardware interrupts are disabled. On some processors, you can disable all interrupts at or below a certain interrupt level and enable all interrupts above that level. On other processors you can simply specify that all interrupts are either enabled or disabled. Details are provided below:

PPC	On PowerPC, x86, MIPS, and ARM processors, you can simply enable or disable all hardware interrupts. You do this by placing either <code>T_ISR</code> or <code>T_NOISR</code> in the <code>mode</code> argument.
x86	
MIPS	
ARM	

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Task already deleted.
0x06	ERR_OBJID	tid incorrect: failed validity check.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x12	ERR_ACTIVE	Task already started.
0x53	ERR_ILLRSC	Task not created from this node.
0x0F	ERR_NOSTK	Task started in user mode with no user mode stack.

1

Notes

1. Once started, the task can compete for the CPU and all other system resources. Thus, it can preempt the calling task if it has a higher priority.
2. t_start() calls the optional user-supplied callout procedure whose address is defined by entry kc_startco in the pSOS+ Configuration Table.

Multiprocessor Considerations

None. A task can only be started from the local node only.

Callable From

- Task

See Also

t_create, t_restart, t_ident

t_suspend Suspends a task indefinitely.

```
#include <psos.h>
unsigned long t_suspend(
    unsigned long tid    /* task identifier */
)
```

Description

This system call suspends execution of a task until a `t_resume()` call is made for the suspended task. The calling task suspends either itself or another task. The `t_suspend()` call prevents the specified task from contending for CPU time but does not directly prevent contention for any other resource. See Note 4.

Arguments

<code>tid</code>	Specifies the task to suspend. If <code>tid</code> equals zero, the calling task suspends itself.
------------------	---

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x05	ERR_OBJDEL	Task was already deleted.
0x06	ERR_OBJID	<code>tid</code> incorrect: failed validity check.
0x07	ERR_OBJTYPE	Object type doesn't match object ID; failed validity check.
0x14	ERR_SUSP	Task was already suspended.

Notes

1. A task that calls `t_suspend()` on itself always returns 0.
2. A suspended task can be deleted.

3. `t_resume()` is the only call that reverses a suspension.
4. A task can be suspended in addition to being blocked. For example, if a task is waiting for a message at a queue when suspension is ordered, suspension continues after a message has been received. For another example, consider a task P that is blocked while it waits for an event. Another task Q decides that P must not run, and therefore Q suspends P. When P receives the event, it must still wait for a resumption before it can be ready to run. On the other hand, if Q resumes P while P is still waiting for its event, P continues to wait for the event.

Multiprocessor Considerations

If `tid` identifies a global task residing on another processor node, the local kernel internally makes an RSC to the remote node to suspend the task.

Callable From

- Task

See Also

`t_resume`

tm_cancel

Cancels an armed timer.

```
#include <psos.h>
unsigned long tm_cancel(
    unsigned long tmid    /* timer identifier */
)
```

Description

This system call enables a task to cancel a timer armed previously by `tm_evafter()`, `tm_evwhen()`, or `tm_evevery()`. The timer must have been armed by the calling task.

Arguments

`tmid` Specifies the timer to cancel.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x4C	ERR_BADTMID	The <code>tmid</code> is invalid.
0x4D	ERR_TMNOTSET	Timer not armed or else already expired.

Notes

When a task with armed timers is restarted or deleted, its timers are automatically cancelled.

Multiprocessor Considerations

None. This call affects only the calling task.

Callable From

- Task

See Also

`tm_evafter`, `tm_every`, `tm_evwhen`

tm_evafter Sends events to the calling task after a specified interval.

```
#include <psos.h>
unsigned long tm_evafter(
    unsigned long ticks,    /* delay */
    unsigned long events,   /* event list */
    unsigned long *tmid     /* timer identifier */
)
```

Description

This system call enables the calling task to arm a timer so that it expires after the specified interval, at which time the pSOS+ kernel internally calls `ev_send()` to send the designated events to this task. Unlike `tm_wkafter()`, `tm_evafter()` does not block the caller. A task can use multiple `tm_evafter()` calls to arm two or more concurrent timers.

The timer interval is specified in system clock ticks. For example, if the system clock frequency is 60 ticks per second and the caller requires a timer to interrupt in 20 seconds, the input specification should be `60x20` (ticks=1200). A timer interval of `n` ticks causes the events to be sent on the `n`th next tick. Because `tm_evafter()` can happen anywhere between two ticks, the actual interval is between `n-1` and `n` ticks.

Arguments

ticks	Specifies the timer interval.
events	Specifies the events to deliver upon expiration of the timer interval. The events are encoded into a long word with bits 31-16 reserved for system use and bits 15 - 0 available for application use.
tmid	Points to the variable where <code>tm_evafter()</code> stores a timer identifier, which can be used if the armed timer must be cancelled.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x4B	ERR_NOTIMERS	Exceeds the maximum number of configured timers.

Notes

1. The maximum interval is $2^{32}-1$ ticks.
2. The timer is counted down by successive `tm_tick()` calls. If no clock or timer is provided, a timer does not expire.
3. A task must call `ev_receive()` explicitly to receive any timer-triggered events (which are like other events in every other way).
4. A task with active timers can be blocked or suspended. In either case, the designated events are sent when the timer expires.
5. When a task with armed timers is restarted or deleted, its timers are automatically cancelled.
6. The number of simultaneously active timers is fixed and defined by the `kc_ntimer` entry in the pSOS+ Configuration Table.

Multiprocessor Considerations

None. This call only affects the calling task.

Callable From

- Task

See Also

`ev_receive`, `ev_send`

tm_evevery Sends events to the calling task at periodic intervals.

```
#include <psos.h>
unsigned long tm_evevery(
    unsigned long ticks,    /* delay */
    unsigned long events,   /* event list */
    unsigned long *tmid     /* timer identifier */
)
```

Description

This system call is similar to `tm_evafter()` except that the armed timer expires periodically instead of once. Events are generated at the specified interval until the timer is cancelled with `tm_cancel()`.

The `tm_evevery()` call provides a drift-free mechanism for performing an operation at periodic intervals. Like `tm_evafter()`, the interval is specified in system clock ticks.

Arguments

<code>ticks</code>	Specifies the periodic interval in system clock ticks.
<code>events</code>	Specifies the events to deliver upon expiration of the timer interval. The events are encoded into a long word with bits 31-16 reserved for system use and bits 15 - 0 available for application use.
<code>tmid</code>	Points to the variable where <code>tm_evevery()</code> stores a timer identifier, which can be used if the armed timer must be cancelled.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x4B	ERR_NOTIMERS	Exceeds the maximum number of configured timers.

Notes

1. The maximum interval is $2^{32}-1$ ticks.
2. A timer is counted down by successive `tm_tick()` calls. If no clock or timer is provided, a timer does not expire.
3. A task must explicitly call `ev_receive()` to receive timer-triggered events (which are like other events in every other way).
4. A task with active timers can be blocked or suspended. In either case, the designated events are sent when the timer expires.
5. When a task with armed timers is restarted or deleted, its timers are automatically cancelled.
6. The number of simultaneously active timers is fixed and defined by the `kc_ntimer` entry in the pSOS+ Configuration Table.

Multiprocessor Considerations

None. This call affects only the calling task.

Callable From

- Task

See Also

`ev_receive`, `ev_send`

tm_evwhen Sends events to the calling task at a specified time.

```
#include <psos.h>
unsigned long tm_evwhen(
    unsigned long date,    /* date of wakeup */
    unsigned long time,    /* time of wakeup */
    unsigned long ticks,   /* ticks at wakeup */
    unsigned long events,  /* event list */
    unsigned long *tmid    /* timer identifier */
)
```

Description

This system call enables the calling task to arm a timer so that it expires at the appointed date and time, whereupon the pSOS+ kernel internally calls `ev_send()` to send the designated events to the task. `tm_evwhen()` does not block the calling task (unlike `tm_wkwhen()`). A task can use multiple `tm_evwhen()` calls to arm two or more concurrent timers.

The `tm_evwhen()` call resembles `tm_evafter()` except that `tm_evwhen()` wakes the caller at an appointed time rather than after a specified interval.

Arguments

`date` Specifies the clock date for event send. `date` is encoded as follows:

Field	Bits
Year, A.D.	31-16
Month (1-12)	15-8
Day (1-31)	7-0

`time` Specifies the clock time for event send. `time` is encoded as follows:

Field	Bits
Hour (0-23)	31-16
Minute (0-59)	15-8
Second (0-59)	7-0

ticks	An optional count that begins after the last second of time has elapsed. This parameter provides a finer resolution of time, if needed.
events	Specifies the events to deliver upon expiration of the timer. The events are encoded into a long word with bits 31-16 reserved for system use and bits 15 - 0 available for application use.
tmid	Points to the variable where tm_evwhen() stores a timer identifier, which can be used if the armed timer must be cancelled.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x47	ERR_NOTIME	System time and date are not yet set.
0x48	ERR_ILLDATE	Date input is out of range.
0x49	ERR_ILLTIME	Time of day input is out of range.
0x4A	ERR_ILLTICKS	Ticks input out of range.
0x4B	ERR_NOTIMERS	Exceeds maximum number of configured timers.
0x4E	ERR_TOOLATE	Too late: input date and time are already past.

Notes

1. A timer is counted down by successive tm_tick() calls. If no clock or timer is provided, a timer does not expire.
2. A timer established by tm_evwhen() is affected by a tm_set() call if that call changes the date and time. If a tm_set() advances the time past a scheduled alarm, it triggers the ev_send() immediately. To detect this situation, the application can check the time when it awakens and compare it with the expected time.
3. A task must explicitly call ev_receive() to receive a timer-triggered event (which is like any event in every other way).

4. A task with active timers can be blocked or suspended. In either case, the designated events are sent when the timer expires.
5. When a task with armed timers is restarted or deleted, its timers are automatically cancelled.
6. The number of simultaneously active timers is fixed and defined by the `kc_ntimer` entry in the pSOS+ Configuration Table.

Multiprocessor Considerations

None. This call affects the calling task only.

Callable From

- Task

See Also

`ev_receive`, `ev_send`

tm_get Obtains the system's current version of the date and time.

```
#include <psos.h>
unsigned long tm_get(
    unsigned long *date,      /* year/month/day */
    unsigned long *time,      /* hour:minute:second */
    unsigned long *ticks      /* ticks */
)
```

Description

This service call returns the system's current version of the date and time-of-day. If the system has no real-time clock, the returned values are meaningless.

Arguments

date	Points to the variable where tm_get() stores the date. date is encoded as follows:	
	Field	Bits
	Year, A.D.	31-16
	Month (1-12)	15-8
	Day (1-31)	7-0
time	Points to the variable where tm_get() stores the time. time is encoded as follows:	
	Field	Bits
	Hour (0-23)	31-16
	Minute (0-59)	15-8
	Second (0-59)	7-0
ticks	Points to the variable where tm_get() stores the number of ticks from the last second of the time argument.	

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x47	ERR_NOTIME	System date and time not set.

Notes

1. The accuracy of the returned date and time depends on the precision of `tm_tick()` activity and the moment that the most recent `tm_set()` call occurred.
2. The algorithm for this call accounts for leap years.

Multiprocessor Considerations

None. This call can only be directed at the local processor node.

Callable From

- Task
- ISR
- KI
- Callout

See Also

`tm_set`, `tm_tick`

tm_set

Sets or resets the system's version of the date and time.

```
#include <psos.h>
unsigned long tm_set(
    unsigned long date,    /* year/month/day */
    unsigned long time,    /* hour:minute:second */
    unsigned long ticks    /* clock ticks */
)
```

1

Description

This service call enables a task either to set or reset the system's version of the date and time. If a meaningful date and time are required, this call should be made after each system restart or power-on. Thereafter, the system maintains the date and time based on incoming `tm_tick()` calls and the expected arrival frequency defined in the pSOS+ Configuration Table.

Arguments

date	Specifies the clock date. date is encoded as follows:	
	Field	Bits
	Year, A.D.	31-16
	Month (1-12)	15-8
	Day (1-31)	7-0
time	Specifies the clock time. time is encoded as follows:	
	Field	Bits
	Hour (0-23)	31-16
	Minute (0-59)	15-8
	Second (0-59)	7-0
ticks	Specifies the number of ticks from the last second of the time argument.	

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x48	ERR_ILLDATE	Date input out of range.
0x49	ERR_ILLTIME	Time input out of range.
0x4A	ERR_ILLTICKS	Ticks input out of range.

Notes

1. This implementation accurately reflects leap years and the current century. For example, the value 0088 means 88 A.D., not 1988 A.D.
2. The pSOS+ kernel maintains a flag that indicates if the system time has been initialized since the last system reboot. Startup clears the flag, and `tm_set()` sets the flag.
3. If the input values are accurate when this call is made, the actual synchronization of the system clock depends on such variables as the execution time of `tm_set()` and the moment it arrives between two ticks. The accuracy is within one or two ticks.
4. `tm_set()` has no effect on tasks that are either timing out or waiting after `tm_wkafter()` or `tm_evafter()` calls because these pause intervals are in clock ticks, not clock time.

Multiprocessor Considerations

None. This call can only be directed at the local processor node.

Callable From

- Task
- ISR

See Also

`tm_get`, `tm_tick`

tm_tick Announces a clock tick to the pSOS+ kernel.

```
#include <psos.h>
unsigned long tm_tick()
```

Description

This system call is used to inform the pSOS+ kernel of the arrival of a new clock tick. The pSOS+ time manager uses it to update its time and date calendar, count down tasks that are timing out, and track a running task's time-slice for round-robin scheduling. Normally, the user's real-time clock ISR calls `tm_tick()`.

The frequency of `tm_tick()` calls is fixed and defined in the pSOS+ Configuration Table as `kc_ticks2sec`. Thus, if the value is 100, the pSOS+ time manager interprets 100 `tm_tick()` calls as one real-time second.

Return Value

This call always returns 0.

Error Codes

None

Notes

1. `tm_tick()` is very fast: it just notifies the system of the arrival of another clock tick. Most other Time Manager actions that can result from this clock tick are postponed until the pSOS+ kernel dispatches and do not run at the clock interrupt level. This improves deterministic system interrupt response.
2. The system accumulates announced ticks when necessary, so no chance exists for an overrun or missed tick. Typically, the accumulation never counts past 1. However, if a system contains one or more lengthy ISRs that respond to high frequency interrupt sources, they can monopolize the CPU enough to prevent the pSOS+ kernel from processing a tick before the next one arrives. In such rare cases, the pSOS+ kernel accumulates the ticks for subsequent accounting.

Multiprocessor Considerations

None. This call can only be directed at the local processor node.

Callable From

- Task
- ISR
- KI
- Callout

See Also

`tm_get`, `tm_set`

tm_wkafter Blocks the calling task and wakes it after a specified interval.

```
#include <psos.h>
unsigned long tm_wkafter(
    unsigned long ticks    /* clock ticks */
)
```

Description

This system call enables the calling task to block unconditionally for a specified interval. This call resembles self-suspension (`t_suspend(0)`), except that `tm_wkafter()` schedules an automatic resumption after the specified time interval has lapsed. The interval is in system clock ticks. For example, if the system clock frequency is 60 ticks per second and the caller requires a pause of 20 seconds, the input specification should be `60x20` (`ticks=1200`).

Arguments

`ticks` Specifies the number of ticks to elapse during the block.

An interval of `n` ticks awakens the calling task on the `n`th next tick. Because `tm_wkafter()` can happen anywhere between two ticks, the actual interval is between `n-1` and `n` ticks.

An interval of 0 ticks has a special function: if no ready tasks have the same priority as the calling (or running) task, the calling task continues. On the other hand, if one or more ready tasks with the same priority as the caller exist, the pSOS+ kernel executes a round-robin by placing the caller behind all ready tasks of the same priority and giving the CPU to one of those tasks. This provides a manual round-robin technique to voluntarily give the CPU to another ready task of the same priority.

Return Value

This call always returns 0.

Error Codes

None.

Notes

1. The maximum interval is $2^{32}-1$ ticks.
2. Each successive `tm_tick()` call counts down the specified delay interval. If no clock or timer is provided, the delay interval does not expire.
3. A delayed task can additionally be suspended, and countdown continues regardless. If not cancelled, suspension continues after expiration.
4. A paused task can be deleted.
5. `tm_set()` calls do not affect a pause established by `tm_wkafter()` because the pause counter is not changed (even if the date and time are changed).

Multiprocessor Considerations

None. This call only affects the calling task.

Callable From

- Task

See Also

`tm_tick`, `tm_wkwhen`

tm_wkwhen Blocks the calling task and wakes it at a specified time.

```
#include <psos.h>
unsigned long tm_wkwhen(
    unsigned long date,    /* year/month/day */
    unsigned long time,    /* hour:minute:second */
    unsigned long ticks    /* clock ticks */
)
```

Description

This call enables the calling task to block unconditionally until a specified time.

The `tm_wkwhen()` call resembles a self-suspension (`t_suspend(0)`), but `tm_wkwhen()` schedules the task to resume at a specified time. This call also resembles `tm_wkafter()`, which awakens the calling task after a specified interval.

Arguments

date	Specifies the clock date. <code>date</code> is encoded as follows:	
	Field	Bits
	Year, A.D.	31-16
	Month (1-12)	15-8
	Day (1-31)	7-0
time	Specifies the clock time. <code>time</code> is encoded as follows:	
	Field	Bits
	Hour (0-23)	31-16
	Minute (0-59)	15-8
	Second (0-59)	7-0
ticks	Specifies the number of ticks within the last second of the <code>time</code> argument.	

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x47	ERR_NOTIME	System time and date not yet set.
0x48	ERR_ILLDATE	Date input out of range.
0x49	ERR_ILLTIME	Time input out of range.
0x4A	ERR_ILLTICKS	Ticks input out of range.
0x4E	ERR_TOOLATE	Too late: input date and time already past.



Notes

1. A `tm_set()` call (which changes the date and time) directly affects the wakeup established by `tm_wkwhen()`. If `tm_set()` advances the time past a scheduled wakeup, it triggers the wakeup immediately. If necessary, the application can detect this situation by checking the time when the task awakens and comparing that time to the expected time.
2. A task can be suspended while it waits for wakeup. In this case, the wait continues: if not cancelled, suspension continues after wakeup.
3. A task can be deleted while it waits for wakeup.

Multiprocessor Considerations

None. This call affects only the calling task.

Callable From

- Task
- ISR

See Also

`tm_tick`, `tm_wkafter`

This chapter provides detailed information on each system call in the pHILE+ component of pSOSystem. The calls are listed alphabetically, with a multipage section of information for each call. Each call's section includes its syntax, the volume types it applies to, a detailed description, its arguments, its return value, and any error codes that it can return. Where applicable, the section also includes the headings "Notes", "Usage", and "See Also". "Notes" contains any important information not specifically related to the call description, "Usage" provides detailed usage information, and "See Also" indicates other calls that have related information.

Structures described in this chapter are also defined in the file <phile.h>. Structures must be word-aligned and must not be packed.

If you need to look up a system call by its functionality, refer to Appendix A, "Tables of System Calls," which lists the calls alphabetically by component and provides a brief description of each call.

The following table shows the file systems that each pHILE+ call supports. If a call supports a particular file system, the table entry is "yes." Otherwise, the table entry is the error message produced when a call is either incorrectly used on a file system or attempted on an unsupported file system. Error codes are described in the call descriptions within this chapter, and also in Appendix B, "Error Codes."

TABLE 2-1 File Systems Supported by pHILE+ Calls

Syscall/ Filesystem	pHILE+	MS-DOS	NFS	CD-ROM
access_f	E_FUNC	E_BADMS	yes	E_BADCD
annex_f	yes	E_BADMS	E_BADNFS	E_BADCD

TABLE 2-1 File Systems Supported by pHILE+ Calls (Continued)

Syscall/ Filesystem	pHILE+	MS-DOS	NFS	CD-ROM
cdmount_vol	E_VALIEN	E_VALIEN	E_MNTED	yes
change_dir	yes	yes	yes	yes
chmod_f	E_FUNC	E_BADMS	yes	E_RO
chown_f	E_FUNC	E_BADMS	yes	E_RO
close_dir	yes	yes	yes	yes
close_f	yes	yes	yes	yes
create_f	yes	yes	yes	E_RO
fchmod_f	E_FUNC	E_BADMS	yes	E_RO
fchown_f	E_FUNC	E_BADMS	yes	E_RO
fstat_f	yes	yes	yes	yes
fstat_vfs	yes	yes	yes	yes
ftruncate_f	yes	yes	yes	E_RO
get_fn	yes	yes	E_BADNFS	yes
init_vol	yes	yes	E_MNTED	E_RO
link_f	E_FUNC	E_BADMS	yes	E_BADCD
lock_f	yes	E_BADMS	E_BADNFS	E_BADCD
lseek_f	yes	yes	yes	yes
lstat_f	E_FUNC	E_BADMS	yes	E_BADCD
make_dir	yes	yes	yes	E_RO
mount_vol	yes	E_VALIEN	E_MNTED	E_VALIEN
move_f	yes	yes	yes	E_RO
nfsmount_vol	E_MNTED	E_MNTED	yes	E_MNTED
open_dir	yes	yes	yes	yes
open_f	yes	yes	yes	yes
open_fn	yes	yes	E_BADNFS	yes

TABLE 2-1 File Systems Supported by pHILE+ Calls (Continued)

Syscall/ Filesystem	pHILE+	MS-DOS	NFS	CD-ROM
pcinit_vol	yes	yes	E_MNTED	E_RO
pcmount_vol	E_VALIEN	yes	E_MNTED	E_VALIEN
read_dir	yes	yes	yes	yes
read_f	yes	yes	yes	yes
read_link	E_FUNC	E_BADMS	yes	E_BADCD
read_vol	yes	yes	E_BADNFS	yes
remove_f	yes	yes	yes	E_RO
stat_f	yes	yes	yes	yes
stat_vfs	yes	yes	yes	yes
symlink_f	E_FUNC	E_BADMS	yes	E_BADCD
sync_vol	yes	yes	E_BADNFS	E_RO
truncate_f	yes	yes	yes	E_RO
unmount_vol	yes	yes	yes	yes
utime_f	E_FUNC	E_BADMS	yes	E_RO
verify_vol	yes	E_VALIEN	E_VALIEN	E_VALIEN
write_f	yes	yes	yes	E_RO
write_vol	yes	yes	E_BADNFS	E_RO

access_f Determines the accessibility of a file.

```
#include <phile.h>
unsigned long access_f(
    char *name,    /* file pathname */
    int mode       /* file mode to check */
)
```

Volume Types

NFS formatted volumes.

Description

access_f() checks the named file for accessibility according to mode.

Arguments

- name Points to a null-terminated pathname of a file to be checked.
- mode Specifies the file mode to check. mode is the result of an OR operation performed on the following constants (defined in <phile.h>.)

Hex	Mnemonic	Description
4	R_OK	Test for read permission.
2	W_OK	Test for write permission.
1	X_OK	Test for execute/search permission.
0	F_OK	Test for presence of file.

Return Value

This system call returns 0 on success, or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2001	E_FUNC	pHILE+ format volume; illegal operation.
0x200C	E_IFN	Illegal pathname.
0x200D	E_NDD	No default directory.
0x2025	E_IDN	Illegal device name.
0x2026	E_BADMS	MS-DOS volume; illegal operation.
0x2051	E_MAXLOOP	Symbolic links nested too deeply.
0x2052	E_EREMOTE	Too many levels of remote in path.
0x2054	E_EIO	A hard error occurred at remote site.
0x2055	E_EACCES	Task does not have permissions.
0x2058	E_ESTALE	Stale NFS file handle.
0x205B	E_ENXIO	No such device or address.
0x205C	E_ENODEV	No such device.
0x2060	E_BADCD	CD-ROM volume; illegal operation.
0x2070	E_EAUTH	RPC authorization is not available.
0x2071	E_ENFS	Portmap failure on the host.
0x2072	E_ETIMEOUT	NFS call timed out.
0x2074	E_ENOAUTHBLK	No RPC authorization blocks available.
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.
0x2077	E_PROBUNAVAIL	Program not available.
0x2078	E_EPROGVERSMISMATCH	Program version mismatched.
0x2079	E_ECANTDECODEARGS	Decode arguments error.
0x207A	E_EUNKNOWNHOST	Unknown host name.
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.
0x207C	E_UNKNOWNPROTO	Unknown protocol.

Hex	Mnemonic	Description
0x207D	E_EINTR	Call interrupted.
0x207E	ERPC	All other RPC errors.

See Also

chmod_f, stat_f, fstat_f

annex_f Allocates contiguous blocks to a file.

```
#include <phile.h>
unsigned long annex_f(
    unsigned long fid,          /* file identifier */
    unsigned long alloc_size,   /* number of blocks to add */
    unsigned long *blkcount     /* number of blocks added */
)
```

Volume Types

pHILE+ formatted volumes.

Description

`annex_f()` extends the physical size of a file on a pHILE+ formatted volume by adding a group of contiguous blocks.

Arguments

<code>fid</code>	Identifies the file.
<code>alloc_size</code>	Specifies the desired number of blocks to add to the file.
<code>blkcount</code>	Points to the variable where <code>annex_f()</code> stores the number of blocks actually allocated. This number can be less than <code>alloc_size</code> , in which case <code>blkcount</code> represents the largest group of contiguous blocks available on the volume.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2015	E_RO	Operation not allowed on read-only system files, directories, or mounted volumes.

Hex	Mnemonic	Description
0x201A	E_FIDBIG	Invalid FID; exceeds maximum.
0x201B	E_FIDOFF	Invalid FID, file is closed.
0x201C	E_ININFULL	Index block is full.
0x201D	E_VFULL	Volume is full.
0x2026	E_BADMS	MS-DOS volume; illegal operation.
0x2050	E_BADNFS	NFS volume; illegal operation.
0x2060	E_BADCD	CD-ROM volume; illegal operation.

Notes

1. `annex_f()` expands the physical size of a file but does not change its logical size or the end-of-file position.
2. `read_f()` and `lseek_f()` calls into annexed blocks are not allowed until the logical length of the file is extended by writing data into the annexed blocks.
3. A volume full error occurs if no blocks can be added to the file.
4. Unless the blocks are merged into the file's last extent, a new extent descriptor is added to the file as a result of an `annex_f()` call.
5. On volumes with separate control and data regions, the pHILE+ file system manager automatically determines the type of block to be annexed (based on the file type.) Directory files receive control blocks, and ordinary files receive data blocks.
6. Annexes to `BITMAP.SYS` and `FLIST.SYS` are not allowed.

See Also

`write_f`, `open_f`, `read_f`, `lseek_f`

cdmount_vol Mounts a CD-ROM volume.

```
#include <phile.h>
unsigned long cdmount_vol(
    char *device,          /* volume name */
    unsigned long sync_mode /* synchronization mode */
)
```

Volume Types

CD-ROM formatted volumes that conform to ISO-9660 specification. Multi-volume sets and interleaved files are not supported.

Description

cdmount_vol() mounts a CD-ROM volume. A volume must be mounted before file operations can be applied to it.

Removable volumes can be mounted and unmounted as required. CD-ROM volumes are read-only.

Arguments

device	Points to the null-terminated name of the volume to be mounted.
sync_mode	Specifies the volume's write synchronization attribute. This attribute is defined in <phile.h> and must be set to the value shown below.
	SM_READ_ONLY Read-only synchronization mode.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2006	E_MNTHFULL	Attempted to mount too many volumes.
0x2007	E_VALIEN	Wrong volume format.
0x2008	E_MNTED	Volume already mounted.
0x2021	E_ILLLDEV	Illegal device (exceeded maximum).
0x2024	E_FMODE	Illegal synchronization mode.
0x2025	E_IDN	Illegal device name.
0x2041	E_BUFSIZE	Buffers not available for block size.
0x2051	E_MAXLOOP	Symbolic links nested too deeply.
0x2061	E_NCDVOL	Configuration Table shows no CD-ROM volume in the system.
0x2062	E_CDMVOL	Cannot support multivolume CD-ROM set.
0x2063	E_CDBSIZE	Volume not made with 2K block size.
0x2064	E_CDFMT	Volume format not ISO-9660 compliant.

Notes

1. A CD-ROM volume does not need volume initialization.
2. The number of volumes that can be mounted simultaneously in the system cannot exceed the pHILE+ Configuration Table parameter `fc_nmount`.
3. The pHILE+ file system manager does not attempt verification or any other way of determining volume ownership. Any task can perform a `cdmount_vol()`. A mounted device does not retain a record of the task that mounted it. Therefore, a volume is not automatically unmounted when the task that mounted it is deleted. This and any other security measures, if desired, should be supported by the user's own layer of software.
4. To enable an application to mount an MS-DOS volume, you must set the mount flag `FC_MSDDOS` in `sys_conf.h`.

See Also

`mount_vol`, `nfsmount_vol`, `pcmount_vol`, `unmount_vol`

change_dir Changes the current directory.

```
#include <phile.h>
unsigned long change_dir(
    char *name    /* directory path */
)
```

Volume Types

All volume types.

Description

change_dir() changes the current directory of the calling task. After change_dir() executes, all relative pathnames used by the calling task are relative to the new current directory.

Arguments

name Points to the null-terminated pathname of the new current directory.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2003	E_BADVOL	Inconsistent data on volume; volume corrupted.
0x200A	E_DMOUNT	Volume not mounted.
0x200B	E_FNAME	Filename not found.
0x200C	E_IFN	Illegal pathname.
0x200D	E_NDD	No default directory.
0x200E	E_FORD	Directory file expected.

Hex	Mnemonic	Description
0x2025	E_IDN	Illegal device name.
0x2051	E_MAXLOOP	Symbolic links nested too deeply.
0x2052	E_EREMOTE	Too many levels of remote in path.
0x2054	E_EIO	A hard error occurred at a remote site.
0x2055	E_EACCESS	Task does not have access permissions.
0x2058	E_ESTALE	Stale NFS file handle.
0x205B	E_ENXIO	No such device or address.
0x205C	E_ENODEV	No such device.
0x2070	E_EAUTH	RPC authorization is not available.
0x2071	E_ENFS	Portmap failure on the host.
0x2072	E_ETIMEOUT	NFS call timed out.
0x2074	E_ENOAUTHBLK	No RPC authorization blocks available.
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.
0x2077	E_PROBUNAVAIL	Program not available.
0x2078	E_EPROGVERSMISMATCH	Program version mismatched.
0x2079	E_ECANTDECODEARGS	Decode arguments error.
0x207A	E_EUNKNOWNHOST	Unknown host name.
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.
0x207C	E_UNKNOWNPROTO	Unknown protocol.
0x207D	E_EINTR	Call interrupted.
0x207E	ERPC	All other RPC errors.

Notes

1. The pHILE+ file system manager does not assume a default current directory. Therefore, if a task uses relative pathnames, it must specify the current directory at least once.
2. The input pathname for the new current directory can be a relative pathname. In this case, it is relative to the current directory before the current directory is changed.
3. The pHILE+ file system manager makes no attempt to verify that the current directory corresponds to the intended entities. For example, if the current directory is deleted or the volume containing the current directory is unmounted, the results of operations by tasks using pathnames relative to the invalid current directory are unpredictable.

See Also

`get_fn()`, `stat_f()`

chmod_f Changes the mode of a named file.

```
#include <phile.h>
unsigned long chmod_f(
    char *name,    /* file pathname */
    int mode       /* new file mode */
)
```

Volume Types

NFS formatted volumes.

Description

chmod_f () changes mode of the named ordinary or directory file.

Arguments

name	Points to a null-terminated pathname of a file.
mode	Specifies the new file mode. mode is the result of an OR operation performed on the following constants (defined in <phile.h>).

Mnemonic	Description
S_ISUID	Set user ID on execution.
S_ISGID	Set group ID on execution.
S_ISVTX	Save text image after execution (sticky bit.)
S_IREAD	Read permission, owner.
S_IWRITE	Write permission, owner.
S_IEXEC	Execute/search permission, owner.
S_IRGRP	Read permission, group.
S_IWGRP	Write permission, group.
S_IXGRP	Execute/search permission, group.
S_IROTH	Read permission, other.
S_IWOTH	Write permission, other.
S_IXOTH	Execute/search permission, other.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2001	E_FUNC	pHILE+ format volume; illegal operation.
0x200C	E_IFN	Illegal pathname.
0x200D	E_NDD	No default directory.
0x2015	E_RO	Operation not allowed on read-only system files, directories, or mounted volumes.
0x2025	E_IDN	Illegal device name.
0x2026	E_BADMS	MS-DOS volume; illegal operation.
0x2051	E_MAXLOOP	Symbolic links nested too deeply.
0x2052	E_EREMOTE	Too many levels of remote in path.
0x2054	E_EIO	A hard error occurred at a remote site.
0x2055	E_EACCES	Task does not have access permissions.
0x2058	E_ESTALE	Stale NFS file handle.
0x205B	E_ENXIO	No such device or address.
0x205C	E_ENODEV	No such device.
0x2060	E_BADCD	CD-ROM volume; illegal operation.
0x2070	E_EAUTH	RPC authorization is not available.
0x2071	E_ENFS	Portmap failure on the host.
0x2072	E_ETIMEOUT	NFS call timed out.
0x2074	E_ENOAUTHBLK	No RPC authorization blocks available.
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.

Hex	Mnemonic	Description
0x2077	E_PROBUNAVAIL	Program not available.
0x2078	E_EPROGVERSMISMATCH	Program version mismatched.
0x2079	E_ECANTDECODEARGS	Decode arguments error.
0x207A	E_EUNKNOWNHOST	Unknown host name.
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.
0x207C	E_UNKNOWNPROTO	Unknown protocol.
0x207D	E_EINTR	Call interrupted.
0x207E	ERPC	All other RPC errors.

See Also

fcntl_f, stat_f, fstat_f, open_f, chown_f, fchown_f

chown_f Changes the owner or group of a named file.

```
#include <phile.h>
unsigned long chown_f(
    char *name,      /* file pathname */
    int owner,        /* new user ID */
    int group         /* new group ID */
)
```

2

Volume Types

NFS formatted volumes.

Description

chown_f () changes the owner and group of a file specified by name.

Arguments

- name Points to a null-terminated pathname of either an ordinary file or a directory file.
- owner Specifies the user ID of the new owner.
- group Specifies the group ID of the new group.

User ID and group ID are UNIX terms used to identify a user and a file access group on a UNIX system.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2001	E_FUNC	pHILE+ format volume; illegal operation.
0x200C	E_IFN	Illegal pathname.

Hex	Mnemonic	Description
0x200D	E_NDD	No default directory.
0x2015	E_RO	Operation not allowed on read-only system files, directories, or mounted volumes.
0x2025	E_IDN	Illegal device name.
0x2026	E_BADMS	MS-DOS volume; illegal operation.
0x2051	E_MAXLOOP	Symbolic links nested too deeply.
0x2052	E_EREMOTE	Too many levels of remote in path.
0x2054	E_EIO	A hard error occurred at a remote site.
0x2055	E_EACCES	Task does not have access permissions.
0x2058	E_ESTALE	Stale NFS file handle.
0x205B	E_ENXIO	No such device or address.
0x205C	E_ENODEV	No such device.
0x2060	E_BADCD	CD-ROM volume; illegal operation.
0x2070	E_EAUTH	RPC authorization is not available.
0x2071	E_ENFS	Portmap failure on the host.
0x2072	E_ETIMEOUT	NFS call timed out.
0x2074	E_ENOAUTHBLK	No RPC authorization blocks available.
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.
0x2077	E_PROBUNAVAIL	Program not available.
0x2078	E_EPROGVERSMISMATCH	Program version mismatched.
0x2079	E_ECANTDECODEARGS	Decode arguments error.
0x207A	E_EUNKNOWNHOST	Unknown host name.
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.
0x207C	E_UNKNOWNPROTO	Unknown protocol.

Hex	Mnemonic	Description
0x207D	E_EINTR	Call interrupted.
0x207E	ERPC	All other RPC errors.

See Also

fchown_f, stat_f, fstat_f, chmod_f, fchmod_f, open_f

close_dir

Closes an open directory file.

```
#include <phile.h>
unsigned long close_dir(
    XDIR *dir    /* NFS directory handle */
)
```

Volume Types

All volume types.

Description

close_dir() closes the connection to a directory specified by the directory handle dir.

Arguments

dir Points to an XDIR structure defined in <phile.h>.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2003	E_BADVOL	Inconsistent data on volume.
0x201A	E_FIDBIG	Invalid FID, exceeds maximum.
0x201B	E_FIDOFF	Invalid FID, file closed.
0x2052	E_EREMOTE	Too many levels of remote in path.
0x2055	E_EACCES	Task does not have access permissions.

See Also

open_dir

close_f Closes an open file connection.

```
#include <phile.h>
unsigned long close_f(
    unsigned long fid    /* file identifier */
)
```

Volume Types

All volume types.

Description

close_f() closes the connection designated by the file identifier fid. If fid is 0, close_f() closes all of the files opened by the calling task. If close_f() terminates the last connection to a file, the file's FCB is deallocated.

Arguments

fid Specifies the file ID of the file connection to be closed.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2003	E_BADVOL	Inconsistent data on volume; volume corrupted.
0x201A	E_FIDBIG	Invalid FID, exceeds maximum.
0x201B	E_FIDOFF	Invalid FID, file closed.

Notes

1. Because the total number of open files is limited for both tasks and the system as a whole, `close_f()` should be used whenever a file connection is no longer needed.
2. If the pREPC+ library is in use, `close_f(0)` should be called only after the pREPC+ call `fclose(0)`. Otherwise, files that the pREPC+ library is using can be unexpectedly closed.
3. If the task has opened one or more NFS files, `close_f(0)` must precede any `close(0)` call to the pNA+ network manager by the same task.

See Also

`open_f`, `open_fn`

create_f Creates a data file.

```
#include <phile.h>
unsigned long create_f(
    char *name,                /* pathname */
    unsigned long expand_unit, /* expansion factor */
    unsigned long mode         /* access mode */
)
```

Volume Types

All volume types (except CD-ROM); however, `expand_unit` is meaningful only on pHILE+ formatted volumes, and `mode` is meaningful only on NFS volumes.

Description

`create_f()` creates a new ordinary file.

Arguments

<code>name</code>	Points to the null-terminated pathname of the file to create.
<code>expand_unit</code>	For pHILE+ formatted volumes only, specifies the number of contiguous blocks to add whenever the file is expanded during a <code>write_f()</code> system call.
<code>mode</code>	For NFS volumes only, specifies the access modes associated with the file, and is the result of an OR operation performed on the following constants (defined in <code><phile.h></code>).

Mnemonic	Description
<code>S_ISUID</code>	Set user ID on execution
<code>S_ISGID</code>	Set group ID on execution
<code>S_IRUSR</code>	Read permission, owner
<code>S_IWUSR</code>	Write permission, owner
<code>S_IXUSR</code>	Execute/search permission, owner
<code>S_IRGRP</code>	Read permission, group
<code>S_IWGRP</code>	Write permission, group

S_IXGRP	Execute/search permission, group
S_IROTH	Read permission, other
S_IWOTH	Write permission, other
S_IXOTH	Execute/search permission, other

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2003	E_BADVOL	Inconsistent data on volume; volume corrupted.
0x200A	E_DMOUNT	Volume not mounted.
0x200B	E_FNAME	Filename not found.
0x200C	E_IFN	Illegal pathname.
0x200D	E_NDD	No default directory.
0x200E	E_FORD	Directory file expected.
0x200F	E_ASIZE	Illegal expansion unit.
0x2010	E_NODE	Null pathname.
0x2011	E_FEXIST	File already exists.
0x2012	E_FLIST	Too many files on volume.
0x2015	E_RO	Operation is not allowed on read-only system files, directories, or mounted volumes.
0x201C	E_ININFULL	Index block is full.
0x201D	E_VFULL	Volume is full.
0x2025	E_IDN	Illegal device name.
0x2051	E_MAXLOOP	Symbolic links are nested too deeply.
0x2052	E_EREMOTE	Too many levels of remote in path.
0x2054	E_EIO	A hard error occurred at remote site.

Hex	Mnemonic	Description
0x2055	E_EACCES	Task does not have access permissions.
0x2057	E_QUOT	Quota exceeded.
0x2058	E_ESTALE	Stale NFS file handle.
0x205B	E_ENXIO	No such device or address.
0x205C	E_ENODEV	No such device.
0x2060	E_BADCD	CD-ROM volume; illegal operation.
0x2070	E_EAUTH	RPC authorization is not available.
0x2071	E_ENFS	Portmap failure on the host.
0x2072	E_ETIMEDOUT	NFS call timed out.
0x2074	E_ENOAUTHBLK	No RPC authorization blocks available.
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.
0x2077	E_PROBUNAVAIL	Program not available.
0x2078	E_EPROGVERSMISMATCH	Program version mismatched.
0x2079	E_ECANTDECODEARGS	Decode arguments error.
0x207A	E_EUNKNOWNHOST	Unknown host name.
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.
0x207C	E_UNKNOWNPROTO	Unknown protocol.
0x207D	E_EINTR	Call interrupted.
0x207E	ERPC	All other RPC errors.

Notes

1. If a file by the same name already exists, `create_f()` fails. An existing file must first be explicitly deleted using `remove_f()` before the same name can be used for a new file.
2. After a `create_f()` call, the new file is empty. Blocks are allocated as data is written.

3. `create_f()` creates only data files. Use `make_dir()` to create a directory file.
4. `create_f()` does not open a file: use an explicit `open_f()`.
5. For pHILE+ formatted volumes, the input parameter `expand_unit` should be considered carefully because it can affect the data access efficiency of the file.

See Also

`remove_f`, `make_dir`, `open_f`

fchmod_f Changes the mode of a file specified by its file identifier.

```
#include <phile.h>
unsigned long fchmod_f(
    unsigned long fid,    /* file identifier */
    int mode              /* new file mode */
)
```

Volume Types

NFS formatted volumes.

Description

`fchmod_f()` functions the same as `chmod_f()` except that `fchmod_f()` changes the mode of a file by its file identifier instead of its pathname. The file identifier is first obtained with `open_f()`.

Arguments

<code>fid</code>	Specifies the file identifier associated with the file.
<code>mode</code>	Specifies the new file mode. <code>mode</code> is the result of an OR operation performed on the following constants (defined in <code><phile.h></code>).
Mnemonic	Description
<code>S_ISUID</code>	Set user ID on execution.
<code>S_ISGID</code>	Set group ID on execution.
<code>S_ISVTX</code>	Save text image after execution (sticky bit).
<code>S_IREAD</code>	Read permission, owner.
<code>S_IWRITE</code>	Write permission, owner.
<code>S_IEXEC</code>	Execute/search permission, owner.
<code>S_IRGRP</code>	Read permission, group.
<code>S_IWGRP</code>	Write permission, group.
<code>S_IXGRP</code>	Execute/search permission, group.

S_IROTH	Read permission, other.
S_IWOTH	Write permission, other.
S_IXOTH	Execute/search permission, other.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2001	E_FUNC	pHILE+ format volume; illegal operation.
0x2015	E_RO	Operation not allowed on read-only system files, directories, or mounted volumes.
0x201A	E_FIDBIG	Invalid file ID; out of range.
0x201B	E_FIDOFF	Invalid file ID; file not open.
0x2023	E_BADFN	Illegal or unused filename.
0x2025	E_IDN	Illegal device name.
0x2026	E_BADMS	MS-DOS volume; illegal operation.
0x2052	E_EREMOTE	Too many levels of remote in path.
0x2054	E_EIO	A hard error occurred at a remote site.
0x2055	E_EACCES	Task does not have access permissions.
0x2058	E_ESTALE	Stale NFS file handle.
0x205B	E_ENXIO	No such device or address.
0x205C	E_ENODEV	No such device.
0x2060	E_BADCD	CD-ROM volume; illegal operation.
0x2070	E_EAUTH	RPC authorization is not available.
0x2071	E_ENFS	Portmap failure on the host.
0x2072	E_ETIMEDOUT	NFS call timed out.

Hex	Mnemonic	Description
0x2074	E_ENOAUTHBLK	No RPC authorization blocks available.
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.
0x2077	E_PROBUNAVAIL	Program not available.
0x2078	E_EPROGVERSMISMATCH	Program version mismatched.
0x2079	E_ECANTDECODEARGS	Decode arguments error.
0x207A	E_EUNKNOWNHOST	Unknown host name.
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.
0x207C	E_UNKNOWNPROTO	Unknown protocol.
0x207D	E_EINTR	Call interrupted.
0x207E	ERPC	All other RPC errors.

See Also

chmod_f, stat_f, fstat_f, open_f, chown_f, fchown_f

fchown_f

Changes the owner or group of a file specified by its file identifier.

```
unsigned long fchown_f(
    unsigned long fid,    /* file identifier */
    int owner,            /* new user ID */
    int group             /* new group ID */
)
```

Volume Types

NFS formatted volumes.

Description

fchown_f() functions the same as chown_f() except it changes the owner or group of a file by its file identifier instead of its pathname. The file identifier is first obtained with open_f().

Arguments

fid	Specifies the file identifier associated with the file.
owner	Specifies the user ID of the new owner.
group	Specifies the group ID of the new group.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2001	E_FUNC	pHILE+ format volume; illegal operation.
0x2015	E_RO	Operation not allowed on read-only system files, directories, or mounted volumes.

Hex	Mnemonic	Description
0x201A	E_FIDBIG	Invalid file ID; out of range.
0x201B	E_FIDOFF	Invalid file ID; file not open.
0x2026	E_BADMS	MS-DOS volume; illegal operation.
0x2052	E_EREMOTE	Too many levels of remote in path.
0x2054	E_EIO	A hard error occurred at a remote site.
0x2055	E_EACCES	Task does not have access permissions.
0x2058	E_ESTALE	Stale NFS file handle.
0x205B	E_ENXIO	No such device or address.
0x205C	E_NODEV	No such device.
0x2060	E_BADCD	CD-ROM volume; illegal operation.
0x2070	E_EAUTH	RPC authorization is not available.
0x2071	E_ENFS	Portmap failure on the host.
0x2072	E_ETIMEOUT	NFS call timed out.
0x2074	E_ENOAUTHBLK	No RPC authorization blocks available.
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.
0x2077	E_PROBUNAVAIL	Program not available.
0x2078	E_EPROGVERSMISMATCH	Program version mismatched.
0x2079	E_ECANTDECODEARGS	Decode arguments error.
0x207A	E_EUNKNOWNHOST	Unknown host name.
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.
0x207C	E_UNKNOWNPROTO	Unknown protocol.
0x207D	E_EINTR	Call interrupted.
0x207E	ERPC	All other RPC errors.

See Also

`chown_f`, `stat_f`, `fstat_f`, `chmod_f`, `fchmod_f`, `open_f`

fstat_f

Obtains the status of a file specified by its file identifier.

```
#include <phile.h>
unsigned long fstat_f(
    unsigned long fid,    /* file identifier */
    struct stat *buf      /* file status */
)
```

2

Volume Types

All volume types.

Description

`fstat_f()` functions the same as `stat_f()` except that `fstat_f()` obtains information about a file by using the file identifier instead of the name. The file identifier is first obtained with either `open_f()` or `open_fn()`.

Arguments

`fid` Specifies the file identifier associated with the file.

`buf` Points to a `stat` structure defined in `<phile.h>`, as follows:

```
struct stat {
    mode_t st_mode;    /* ownership/protection */
    ino_t st_ino;      /* file ID */
    dev_t st_dev;      /* device ID where the volume resides */
    dev_t st_rdev;     /* device ID, for character or
                       * block special files only */
    nlink_t st_nlink;  /* number of hard links to the file */
    uid_t st_uid;      /* user ID */
    gid_t st_gid;      /* group ID */
    off_t st_size;     /* total size of file, in bytes */
    time_t st_atime;   /* file last access time */
    time_t st_mtime;   /* file last modify time */
    time_t st_ctime;   /* file last status change time */
    long st_blksize;   /* optimal block size for I/O ops */
    long st_blocks;    /* file size in blocks */
};
```

This structure cannot be packed. `mode_t`, `ino_t`, `dev_t`, `nlink_t`, `uid_t`, `gid_t`, `off_t`, and `time_t` are defined as unsigned long in `<phile.h>`.

The following differences exist for local file systems (pHILE+, MS-DOS, and CD-ROM):

```
rdev = dev, nlink = 1, uid = 0, gid = 0, atime = ctime
= mtime
```

The status information word `st_mode` consists of the following bits:

<code>_IFMT</code>	<code>0170000</code>	<code>/* type of file */</code>
<code>_IFIFO</code>	<code>0010000</code>	<code>/* fifo special */</code>
<code>_IFCHR</code>	<code>0020000</code>	<code>/* character special */</code>
<code>_IFDIR</code>	<code>0040000</code>	<code>/* directory */</code>
<code>_IFBLK</code>	<code>0060000</code>	<code>/* block special */</code>
<code>_IFREG</code>	<code>0100000</code>	<code>/* regular file */</code>
<code>_IFLNK</code>	<code>0120000</code>	<code>/* symbolic link */</code>
<code>_IFSOCK</code>	<code>0140000</code>	<code>/* socket */</code>
<code>S_ISUID</code>	<code>0004000</code>	<code>/* set user ID on execution */</code>
<code>S_ISGID</code>	<code>0002000</code>	<code>/* set group ID on execution */</code>
<code>S_ISVTX</code>	<code>0001000</code>	<code>/* save swapped text even after use */</code>
<code>S_IRUSR</code>	<code>0000400</code>	<code>/* read permission, owner */</code>
<code>S_IWUSR</code>	<code>0000200</code>	<code>/* write permission, owner */</code>
<code>S_IXUSR</code>	<code>0000100</code>	<code>/* execute/search permission, owner */</code>
<code>S_IRGRP</code>	<code>0000040</code>	<code>/* read permission, group */</code>
<code>S_IWGRP</code>	<code>0000020</code>	<code>/* write permission, group */</code>
<code>S_IXGRP</code>	<code>0000010</code>	<code>/* execute/search permission, group */</code>
<code>S_IROTH</code>	<code>0000004</code>	<code>/* read permission, other */</code>
<code>S_IWOTH</code>	<code>0000002</code>	<code>/* write permission, other */</code>
<code>S_IXOTH</code>	<code>0000001</code>	<code>/* execute/search permission, other */</code>

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x201A	E_FIDBIG	Invalid file ID; out of range.
0x201B	E_FIDOFF	Invalid file IDE; file not open.
0x2052	E_EREMOTE	Too many levels of remote in path.
0x2054	E_EIO	A hard error occurred at a remote site.
0x2055	E_EACCES	Task does not have access permissions.
0x2058	E_ESTALE	Stale NFS file handle.
0x205B	E_ENXIO	No such device or address.
0x205C	E_ENODEV	No such device.
0x2070	E_EAUTH	RPC authorization is not available.
0x2071	E_ENFS	Portmap failure on the host.
0x2072	E_ETIMEDOUT	NFS call timed out.
0x2074	E_ENOAUTHBLK	No RPC authorization blocks available.
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.
0x2077	E_PROBUNAVAIL	Program not available.
0x2078	E_EPROGVERSMISMATCH	Program version mismatched.
0x2079	E_ECANTDECODEARGS	Decode arguments error.
0x207A	E_EUNKNOWNHOST	Unknown host name.
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.
0x207C	E_UNKNOWNPROTO	Unknown protocol.
0x207D	E_EINTR	Call interrupted.
0x207E	ERPC	All other RPC errors.

See Also

`stat_f`, `chmod_f`, `fchmod_f`, `chown_f`, `fchown_f`, `link_f`, `read_f`,
`read_link`, `truncate_f`, `ftruncate_f`, `remove_f`, `utime_f`, `write_f`

fstat_vfs	Obtains statistics about a mounted volume specified by a file identifier.
------------------	---

```
#include <phile.h>
unsigned long fstat_vfs(
    unsigned long fid,      /* file identifier */
    struct statvfs *buf     /* volume statistics */
)
```

Volume Types

All volumes.

Description

`fstat_vfs()` functions the same as `stat_vfs()` except that `fstat_vfs()` obtains the statistics about a volume by using the file identifier instead of the pathname. The file identifier is first obtained with either an `open_f()` or an `open_fn()` call to any file in the volume.

Arguments

<code>file</code>	Specifies the file identifier of the file, which can be any file within the mounted volume.
-------------------	---

buf **Points to a statvfs structure defined in <phile.h>, as follows:**

```
typedef struct {
    long val[2];
} fsid_t;

struct statvfs {
    unsigned long f_bsize;      /* preferred volume block size */
    unsigned long f_frsize;     /* fundamental volume block size */
    unsigned long f_blocks;     /* total number of blocks */
    unsigned long f_bfree;      /* total number of free blocks */
    unsigned long f_bavail;     /* free blocks available to
    * non-superuser */
    unsigned long f_files;      /* total # of file nodes
    * (pHILE+ files only) */
    unsigned long f_ffree;      /* reserved (not supported) */
    unsigned long f_favail;     /* reserved (not supported) */
    fsid_t f_fsid;              /* reserved (not supported) */
    char f_basetype[16];        /* reserved (not supported) */
    unsigned long f_flag;        /* reserved (not supported) */
    unsigned long f_namemax;     /* reserved (not supported) */
    char f_fstr[32];             /* reserved (not supported) */
    unsigned long f_fstype;      /* file system type number */
    unsigned long f_filler[15]; /* reserved (not supported) */
};
```

This structure cannot be packed. Currently, the fields `f_ffree`, `f_favail`, `f_fsid`, `f_basetype`, `f_flag`, `f_namemax`, `f_fstr` and `f_filler` are reserved and do not have values. For all volumes except pHILE+ format, the field `f_files` is unused.

The field `f_fstype` identifies the type of file system format. The values in <phile.h> are given below:

<code>FSTYPE_PHILE</code>	pHILE+ format volume
<code>FSTYPE_PCDOS</code>	MS-DOS format volume
<code>FSTYPE_CDROM</code>	CD-ROM format volume
<code>FSTYPE_NFS</code>	Client NFS volume

The return value for all unsupported fields is 0.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x201A	E_FIDBIG	Invalid file ID; out of range.
0x201B	E_FIDOFF	Invalid file ID; file not open.
0x2052	E_REMOTE	Too many levels of remote in path.
0x2054	E_EIO	A hard error occurred at a remote site.
0x2055	E_EACCES	Task does not have access permissions.
0x2058	E_ESTALE	Stale NFS file handle.
0x205B	E_ENXIO	No such device or address.
0x205C	E_ENODEV	No such device.
0x2070	E_EAUTH	RPC authorization is not available.
0x2071	E_ENFS	Portmap failure on the host.
0x2072	E_ETIMEOUT	NFS call timed out.
0x2074	E_ENOAUTHBLK	No RPC authorization blocks available.
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.
0x2077	E_PROBUNAVAIL	Program not available.
0x2078	E_EPROGVERSMISMATCH	Program version mismatched.
0x2079	E_ECANTDECODEARGS	Decode arguments error.
0x207A	E_EUNKNOWNHOST	Unknown host name.
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.
0x207C	E_UNKNOWNPROTO	Unknown protocol.
0x207D	E_EINTR	Call interrupted.
0x207E	ERPC	All other RPC errors.

See Also

`stat_vfs`

ftruncate_f	Changes the size of a file specified by its file identifier.
--------------------	--

```
#include <phile.h>
unsigned long ftruncate_f(
    unsigned long fid,      /* file identifier */
    unsigned long length    /* file size in bytes */
)
```

Volume Types

pHILE+, MS-DOS, and NFS formatted volumes.

Description

`ftruncate_f()` functions the same as `truncate_f()` except that `ftruncate_f()` changes the size of a file by using the file identifier instead of the pathname. The file identifier is first obtained with either `open_f()` or `open_fn()`. Unlike `annex_f()`, this system call changes both the logical and the physical file size. (`annex_f()` changes only the physical file size.)

On pHILE+ or MS-DOS volumes, the file must have been opened only once, that is no other task has it open and the calling task has opened it only once. If this is violated, the error `E_FOPEN` is returned.

On pHILE+ or MS-DOS volumes, if the file is truncated shorter than its L_ptr, the L_ptr is changed to the new end-of-file.

Arguments

<code>fid</code>	Specifies the file identifier associated with the file.
<code>length</code>	Specifies the new file size. If the file was previously longer than <code>length</code> , the extra bytes are truncated. If it was shorter, the bytes between the old and new lengths are filled with 0's.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2001	E_FUNC	Invalid function number.
0x2003	E_BADVOL	Inconsistent data on volume; volume corrupted.
0x2015	E_RO	Operation not allowed on read-only system files, directories, or mounted volumes.
0x201A	E_FIDBIG	Invalid file ID; out of range.
0x201B	E_FIDOFF	Invalid file ID; file not open.
0x201C	E_ININFULL	Index block full.
0x201D	E_VFULL	Volume is full. (This cannot happen on NFS formatted volumes.)
0x2022	E_LOCKED	Data is locked.
0x2052	E_EREMOTE	Too many levels of remote in path.
0x2054	E_EIO	A hard error occurred at a remote site.
0x2055	E_EACCES	Task does not have access permissions.
0x2057	E_EQUOT	Quota exceeded.
0x2058	E_ESTALE	Stale NFS file handle.
0x205B	E_ENXIO	No such device or address.
0x205C	E_ENODEV	No such device.
0x2060	E_BADCD	CD-ROM volume; illegal operation.
0x2070	E_EAUTH	RPC authorization is not available.
0x2071	E_ENFS	Portmap failure on the host.
0x2072	E_ETIMEOUT	NFS call timed out.
0x2074	E_ENOAUTHBLK	No RPC authorization blocks available.
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.

Hex	Mnemonic	Description
0x2077	E_PROBUNAVAIL	Program not available.
0x2078	E_EPROGVERSMISMATCH	Program version mismatched.
0x2079	E_ECANTDECODEARGS	Decode arguments error.
0x207A	E_EUNKNOWNHOST	Unknown host name.
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.
0x207C	E_UNKNOWNPROTO	Unknown protocol.
0x207D	E_EINTR	Call interrupted.
0x207E	ERPC	All other RPC errors.

See Also

truncate_f, open_f, open_fn

get_fn Obtains the number of a file.

```
#include <phile.h>
unsigned long get_fn(
    char *name,          /* filename */
    unsigned long *fn     /* file number */
)
```

Volume Types

pHILE+, MS-DOS, and CD-ROM formatted volumes.

Description

get_fn() returns the file number associated with a file. The file number can then be used with an open_fn() call or as part of an absolute filename.

Arguments

name	Points to the null-terminated pathname of the file.
fn	Points to the variable where get_fn() stores the file number.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2003	E_BADVOL	Inconsistent data on volume; volume corrupted.
0x200A	E_DMOUNT	Volume not mounted.
0x200B	E_FNAME	Filename not found.
0x200C	E_IFN	Illegal pathname.
0x200D	E_NDD	No default directory.

Hex	Mnemonic	Description
0x200E	E_FORD	Directory file expected.
0x2025	E_IDN	Illegal device name.
0x2050	E_BADNFS	NFS volume; illegal operation.

Usage

One usage of `get_fn()` is to get the current directory. In the example below, the current directory's FN is returned in `dir_fn`.

```
unsigned long dir_fn;    /*current directory */
if (get_fn(".", &dir_fn) != 0) {
    /*Insert some error processing here */;
}
```

The pseudocode below shows how to get the current directory's full pathname, rather than just the FN.

1. Get the directory's FN in `dir_fn` with `get_fn(".")` as above.
2. Open the parent directory ("`..`") with `open_dir()`.
3. Search the parent directory for the directory entry of the current directory.
 - a. Read a directory entry with `read_dir()`.
 - b. Compare the directory entry's `d_filno` with the FN of the current directory.
 - c. Repeat steps a and b until they match.
 - d. Remember the matching directory entry's `d_name`. It is the last component of the current directory's `pathname`.
4. Close the open directory with `close_dir()`.
5. Repeat steps 1-4 for the parent directory of the current directory, the grandparent of the current directory, etc., until reaching the root directory. The root directory is reached when either `get_fn()` of the parent directory is an error, or `get_fn()` of the parent directory is the same as `get_fn()` of the directory.

6. The answer is the concatenation of all the components found in step 3 d from last to first.

As stated above, `get_fn()` is available for local volumes only (not NFS volumes.)

To obtain not only the current directory, but also the current device, see `stat_f()`.

See Also

`open_fn`, `stat_f()`

init_vol Initializes a pHILE+ formatted volume.

```
#include <phile.h>
unsigned long init_vol(
    char *device,                /* volume name */
    struct INIT_VOL_PARAMS *params, /* parameters */
    void *scratchbuf            /* scratch buffer */
)
```

2

Volume Types

pHILE+ formatted volumes.

Description

`init_vol()` initializes a pHILE+ formatted volume with user-supplied parameters. `init_vol()` performs a logical format of the volume, setting up the necessary control structures and other information needed by the pHILE+ file system manager for subsequent file operations on the volume. A volume must be initialized before it can be mounted.

After a volume has been initialized, `init_vol()` can be used to quickly delete all data on the volume.

`init_vol()` can be used for the first initialization of a volume (see Note 4).

Arguments

`device` Points to the null-terminated volume name.

params Points to an instance of the `init_vol_params` structure, which contains parameters used to initialize the volume. This structure is defined in `<phile.h>` as follows:

```
typedef struct init_vol_params {
    char        volume_label[12]; /* volume label */
    unsigned long volume_size;     /* number blocks in volume */
    unsigned long num_of_file_descriptors;
                                /* number descriptors in
                                * FLIST */
    unsigned long starting_bitmap_block_number;
                                /* first BITMAP block */
    unsigned long start_data_block_number;
                                /* first data block */
} INIT_VOL_PARAMS;
```

This structure cannot be packed. The fields of the `init_vol_params` structure are described below:

volume_label Contains a 12-byte volume label. The pHILE+ file system manager copies the label to the volume's `ROOTBLOCK` but does not use it. (The volume label is not the volume name. The volume name contains the volume's major and minor device numbers.)

volume_size The number of blocks on the volume. For example, a value of 5000 indicates the volume contains blocks 0 - 4999.

num_of_file_descriptors The number of file descriptors in the volume's `FLIST`. This is the number of files that can be created on the volume.

starting_bitmap_block_number The starting block for the volume's `BITMAP`.

start_data_block_number The starting block for the volume's data blocks. The pHILE+ file system manager requires this parameter to be a multiple of 8.

scratchbuf

Points to a buffer that is used temporarily by the pHILE+ file system manager during initialization. The scratch buffer must be the size of a pHILE+ block. The pHILE+ Configuration Table parameter `fc_logbsize` (in the `sys_conf.h` file) determines this block size.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2003	E_BADVOL	Inconsistent data on volume; volume corrupted.
0x2005	E_VINITPAR	Illegal parameters to <code>init_vol()</code> .
0x2008	E_MNTED	Volume is already mounted.
0x2021	E_ILLDEV	Illegal device (exceeds maximum.)
0x2025	E_IDN	Illegal device name.

Notes

1. All data stored on the volume is lost by execution of this call.
2. The volume's media must have been properly hardware formatted before this call is executed.
3. A mounted volume cannot be initialized.
4. `init_vol()` can be used for the first initialization of a volume. It receives all the information it needs in its parameters.
5. The pHILE+ file system manager stores the volume's label and time of initialization in the volume's rootblock, but it does not use this information. The user decides how to use this information, which can be examined by using `read_vol()` to read the rootblock (block 2) directly.
6. The starting block of the bitmap also determines the starting block of the `FLIST`, since the `FLIST` immediately follows the bitmap.

See Also

mount_vol, pcinit_vol

link_f Creates a hard link between two files on the same volume.

```
unsigned long link_f(  
    char *name1,      /* an existing filename */  
    char *name2       /* a new directory entry to be created */  
)
```

2

Volume Types

NFS formatted volumes.

Description

link_f() makes a hard link from name2 to name1. This increments the link count for the file (see stat_f()). After this call, name1 and name2 are two alternate names for the same file. Both files must be on the same volume.

Arguments

- name1 Points to a null-terminated pathname of an existing file. Must not refer to a directory.
- name2 Points to a null-terminated pathname of a directory entry to be created.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2001	E_FUNC	pHILE+ volume; illegal operation.
0x200C	E_IFN	Illegal pathname.
0x200D	E_NDD	No default directory.
0x2015	E_RO	Requested operation not allowed on this file.

Hex	Mnemonic	Description
0x2025	E_IDN	Illegal device name.
0x2026	E_BADMS	MS-DOS volume; illegal operation.
0x2051	E_MAXLOOP	Symbolic links nested too deeply.
0x2052	E_EREMOTE	Too many levels of remote in path.
0x2054	E_EIO	A hard error occurred at a remote site.
0x2055	E_EACCES	Task does not have access permissions.
0x2057	E_EQUOT	Quota exceeded.
0x2058	E_ESTALE	Stale NFS file handle.
0x2059	E_XLINK	Can't close link.
0x205B	E_ENXIO	No such device or address.
0x205C	E_ENODEV	No such device.
0x2060	E_BADCD	CD-ROM volume; illegal operation.
0x2070	E_EAUTH	RPC authorization is not available.
0x2071	E_ENFS	Portmap failure on the host.
0x2072	E_ETIMEOUT	NFS call timed out.
0x2074	E_ENOAUTHBLK	No RPC authorization blocks available.
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.
0x2077	E_PROBUNAVAIL	Program not available.
0x2078	E_EPROGVERSMISMATCH	Program version mismatched.
0x2079	E_ECANTDECODEARGS	Decode arguments error.
0x207A	E_EUNKNOWNHOST	Unknown host name.
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.
0x207C	E_UNKNOWNPROTO	Unknown protocol.
0x207D	E_EINTR	Call interrupted.
0x207E	ERPC	All other RPC errors.

See Also

`symlink_f`, `remove_f`

lock_f

Locks or unlocks part or all of an open file.

```
#include <phile.h>
unsigned long lock_f(
    unsigned long fid,          /* file identifier */
    unsigned long startpos,     /* starting lock position */
    unsigned long bcount       /* number of bytes to lock */
)
```

Volume Types

pHILE+ formatted volumes.

Description

`lock_f()` locks or unlocks part or all of an open file. Following a `lock_f()` system call, only the task that set the lock can access the locked bytes and only through the connection (the file identifier) used to set the lock.

A `lock_f()` call replaces any previous locks it set through the same connection with the new lock. Thus, only one lock per connection can be set. `lock_f()` with `bcount = 0` is used to remove a lock.

A file can have as many locks as it has connections if the locks do not overlap. If a task attempts to lock a region already locked through a different connection, an error is returned, even if the two connections are from the same task.

Arguments

<code>fid</code>	Specifies the file identifier of the file to lock.
<code>startpos</code>	Specifies the starting byte of the locked region.
<code>bcount</code>	Specifies the length of the locked region in bytes.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2015	E_RO	Requested operation not allowed on this file.
0x201A	E_FIDBIG	Invalid FID; exceeds maximum.
0x201B	E_FIDOFF	Invalid FID; file closed.
0x2022	E_LOCKED	Data locked.
0x2026	E_BADMS	MS-DOS volume; illegal operation.
0x2050	E_BADNFS	NFS volume; illegal operation.
0x2060	E_BADCD	CD-ROM volume; illegal operation.

Notes

1. `lock_f()` enables the locked region to begin and/or end beyond the current logical or physical end of the file. In such cases, new data that is appended to the file in the locked region becomes locked.
2. `lock_f()` does not move the `L_ptr`.
3. When initially opened, a file connection has no locks.
4. When a connection to a file is closed, any lock it has on the file is removed.
5. A locked region of a file denies read, write, and truncate access to it by any other file connection. However, `annex_f()`, which expands a file's physical size without changing its logical size, is allowed.
6. Directory and system files cannot be locked.

See Also

`annex_f`

lseek_f Repositions for read or write within an open file.

```
#include <phile.h>
unsigned long lseek_f(
    unsigned long fid,          /* file identifier */
    unsigned long position,     /* relative seek vector */
    long offset,               /* offset */
    unsigned long *old_lptr     /* previous L_ptr */
)
```

Volume Types

All volume types.

Description

`lseek_f()` repositions the `L_ptr` associated with an open file connection. Each file connection has its own `L_ptr`, and it points to the next byte to be read or written in the file. Repositioning can be specified relative to the beginning of the file, the current `L_ptr`, or the end of the file.

Arguments

<code>fid</code>	Specifies the file identifier associated with the file.								
<code>position</code>	Defines how to reposition <code>L_ptr</code> and must have one of the following values: <table> <tr> <th>Value</th><th>Meaning</th></tr> <tr> <td>0</td><td>Offset from beginning of file</td></tr> <tr> <td>1</td><td>Offset from current <code>L_ptr</code></td></tr> <tr> <td>2</td><td>Offset from end of file</td></tr> </table>	Value	Meaning	0	Offset from beginning of file	1	Offset from current <code>L_ptr</code>	2	Offset from end of file
Value	Meaning								
0	Offset from beginning of file								
1	Offset from current <code>L_ptr</code>								
2	Offset from end of file								
<code>offset</code>	Specifies the number of bytes to move <code>L_ptr</code> . A negative offset moves <code>L_ptr</code> backwards.								
<code>old_lptr</code>	Points to the variable where <code>lseek_f()</code> stores the previous value of the <code>L_ptr</code> .								

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x200A	E_DMOUNT	Volume not mounted.
0x201A	E_FIDBIG	Invalid FID; exceeds maximum.
0x201B	E_FIDOFF	Invalid FID, file closed.
0x201E	E_BADPOS	Illegal position parameter.
0x201F	E_EOF	Seek past end of file.
0x2052	E_EREMOTE	Too many levels of remote in path.
0x2054	E_EIO	A hard error happened at remote site.
0x2055	E_EACCES	Task does not have access permissions.
0x2058	E_ESTALE	Stale NFS file handle.
0x205B	E_ENXIO	No such device or address.
0x205C	E_ENODEV	No such device.
0x2070	E_EAUTH	RPC authorization is not available.
0x2071	E_ENFS	Portmap failure on the host.
0x2072	E_ETIMEDOUT	NFS call timed out.
0x2074	E_ENOAUTHBLK	No RPC authorization blocks available.
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.
0x2077	E_PROBUNAVAIL	Program not available.
0x2078	E_EPROGVERSMISMATCH	Program version mismatched.
0x2079	E_ECANTDECODEARGS	Decode arguments error.
0x207A	E_EUNKNOWNHOST	Unknown host name.
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.

Hex	Mnemonic	Description
0x207C	E_UNKNOWNPROTO	Unknown protocol.
0x207D	E_EINTR	Call interrupted.
0x207E	ERPC	All other RPC errors.

Usage

`lseek_f()` can be used to determine the current logical size of a file, as in this example:

```
lseek_f(fid, 2, 0, &oldptr)
lseek_f(fid, 0, oldptr, &filesize)
```

The first call seeks to the end-of-file and saves the original position. The second call restores the original position and obtains the end-of-file position. The end-of-file position is also the file's logical size.

Notes

1. A separate `L_ptr` is associated with each file connection. `lseek_f()` affects only the `L_ptr` associated with the specified file descriptor (`fid`).
2. Because `L_ptr` is unsigned, positioning it before the start of the file results in a seek past end-of-file error.
3. Because `L_ptr` cannot be moved beyond the end of the file, it is not possible to create a file with holes in it.

See Also

`read_f`, `write_f`

lstat_f

Gets the status of a symbolically linked file.

```
#include <phile.h>
unsigned long lstat_f(
    char *name,          /* file pathname */
    struct stat *buf      /* file status */
)
```

2

Volume Types

NFS volumes.

Description

`lstat_f()` is like `stat_f()` except when the named file is a symbolic link. For a symbolic link, `lstat_f()` returns information about the link file, and `stat_f()` returns information about the file to which the link refers.

Arguments

name Points to the null-terminated pathname of a file.

buf Points to a `stat` structure defined in `<phile.h>`, as follows:

```
struct stat {
    mode_t  st_mode;      /* ownership/protection */
    ino_t   st_ino;       /* file ID */
    dev_t   st_dev;       /* dev ID where the volume resides */
    dev_t   st_rdev;      /* dev ID for character or block
                          * special files only */
    nlink_t st_nlink;     /* number of hard links to the file */
    uid_t   st_uid;       /* user ID */
    gid_t   st_gid;       /* group ID */
    off_t   st_size;      /* total size of file, in bytes */
    time_t  st_atime;     /* file last access time */
    time_t  st_mtime;     /* file last modify time */
    time_t  st_ctime;     /* file last status change time */
    long    st_blksize;   /* optimal block size for I/O ops */
    long    st_blocks;    /* file size in blocks */
};
```

This structure cannot be packed. No time zone is associated with the time values.

mode_t, ino_t, dev_t, nlink_t, uid_t, gid_t, off_t, and time_t are defined as unsigned long in <phile.h>.

The status information word st_mode consists of the following bits:

S_IFMT	0170000	/* type of file */
S_IFIFO	0010000	/* fifo special */
S_IFCHR	0020000	/* character special */
S_IFDIR	0040000	/* directory */
S_IFBLK	0060000	/* block special */
S_IFREG	0100000	/* regular file */
S_IFLNK	0120000	/* symbolic link */
S_IFSOCK	0140000	/* socket */
S_ISUID	0004000	/* set user ID on execution */
S_ISGID	0002000	/* set group ID on execution */
S_ISVTX	0001000	/* save swapped text even after use */
S_IRREAD	0000400	/* read permission, owner */
S_IWRITE	0000200	/* write permission, owner */
S_IEXEC	0000100	/* execute/search permission, owner */
S_IRGRP	0000040	/* read permission, group */
S_IWGRP	0000020	/* write permission, group */
S_IXGRP	0000010	/* execute/search permission, group */
S_IROTH	0000004	/* read permission, other */
S_IWOTH	0000002	/* write permission, other */
S_IXOTH	0000001	/* execute/search permission, other */

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2001	E_FUNC	pHILE+ volume; illegal operation.
0x200C	E_IFN	Illegal pathname.
0x200D	E_NDD	No default directory.
0x2025	E_IDN	Illegal device name.
0x2026	E_BADMS	MS-DOS volume; illegal operation.
0x2051	E_MAXLOOP	Symbolic links nested too deeply.
0x2052	E_EREMOTE	Too many levels of remote in path.
0x2054	E_EIO	A hard error occurred at a remote site.
0x2055	E_EACCES	Task does not have access permissions.
0x2058	E_ESTALE	Stale NFS file handle.
0x205B	E_ENXIO	No such device or address.
0x2060	E_BADCD	CD-ROM volume; illegal operation.
0x205C	E_ENODEV	No such device.
0x2060	E_BADCD	CD-ROM volume; illegal operation.
0x2070	E_EAUTH	RPC authorization is not available.
0x2071	E_ENFS	Portmap failure on the host.
0x2072	E_ETIMEDOUT	NFS call timed out.
0x2074	E_ENOAUTHBLK	No RPC authorization blocks are available.
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.
0x2077	E_PROBUNAVAIL	Program not available.
0x2078	E_EPROGVERSMISMATCH	Program version mismatched.
0x2079	E_ECANTDECODEARGS	Decode arguments error.
0x207A	E_EUNKNOWNHOST	Unknown host name.

Hex	Mnemonic	Description
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.
0x207C	E_UNKNOWNPROTO	Unknown protocol.
0x207D	E_EINTR	Call interrupted.
0x207E	ERPC	All other RPC errors.

See Also

`symlink_f`, `stat_f`, `fstat_f`

make_dir Creates a directory file.

```
#include <phile.h>
unsigned long make_dir(
    char *name,           /* directory pathname */
    unsigned long mode    /* access permissions */
)
```

Volume Types

All volume types, except CD-ROM; however, mode is only meaningful for NFS volumes.

Description

make_dir() creates a new directory file.

Arguments

name	Points to the null-terminated pathname of the directory file to create.																				
mode	For NFS volumes only, specifies the access modes associated with the file and is the result of an OR operation performed on the following constants (defined in <phile.h>):																				
	<table><tr><th>Mnemonic</th><th>Description</th></tr><tr><td>S_ISUID</td><td>Set user ID on execution.</td></tr><tr><td>S_ISGID</td><td>Set group ID on execution.</td></tr><tr><td>S_IRUSR</td><td>Read permission, owner.</td></tr><tr><td>S_IWUSR</td><td>Write permission, owner.</td></tr><tr><td>S_IXUSR</td><td>Execute/search permission, owner.</td></tr><tr><td>S_IRGRP</td><td>Read permission, group.</td></tr><tr><td>S_IWGRP</td><td>Write permission, group.</td></tr><tr><td>S_IXGRP</td><td>Execute/search permission, group.</td></tr><tr><td>S_IROTH</td><td>Read permission, other.</td></tr></table>	Mnemonic	Description	S_ISUID	Set user ID on execution.	S_ISGID	Set group ID on execution.	S_IRUSR	Read permission, owner.	S_IWUSR	Write permission, owner.	S_IXUSR	Execute/search permission, owner.	S_IRGRP	Read permission, group.	S_IWGRP	Write permission, group.	S_IXGRP	Execute/search permission, group.	S_IROTH	Read permission, other.
Mnemonic	Description																				
S_ISUID	Set user ID on execution.																				
S_ISGID	Set group ID on execution.																				
S_IRUSR	Read permission, owner.																				
S_IWUSR	Write permission, owner.																				
S_IXUSR	Execute/search permission, owner.																				
S_IRGRP	Read permission, group.																				
S_IWGRP	Write permission, group.																				
S_IXGRP	Execute/search permission, group.																				
S_IROTH	Read permission, other.																				

S_IWOTH	Write permission, other.
S_IXOTH	Execute/search permission, other.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2001	E_FUNC	Invalid function number.
0x2003	E_BADVOL	Inconsistent data on volume; volume corrupted.
0x200A	E_DMOUNT	Volume not mounted.
0x200B	E_FNAME	Filename not found.
0x200C	E_IFN	Illegal pathname.
0x200D	E_NDD	No default directory.
0x200E	E_FORD	Directory file expected.
0x2010	E_NODE	Null pathname.
0x2011	E_FEXIST	File already exists.
0x2012	E_FLIST	Too many files on volume.
0x2015	E_RO	Operation not allowed on read-only system files, directories, or mounted volumes.
0x201C	E_ININFULL	Index block full.
0x201D	E_VFULL	Volume full.
0x2025	E_IDN	Illegal device name.
0x2051	E_MAXLOOP	Symbolic links nested too deeply.
0x2052	E_EREMOTE	Too many levels of remote in path.
0x2054	E_EIO	A hard error happened at remote site.
0x2055	E_EACCES	Task does not have access permissions.

Hex	Mnemonic	Description
0x2057	E_EQUOT	Quota exceeded.
0x2058	E_ESTALE	Stale NFS file handle.
0x205B	E_ENXIO	No such device or address.
0x205C	E_ENODEV	No such device.
0x2060	E_BADCD	CD-ROM volume; illegal operation.
0x2070	E_EAUTH	RPC authorization is not available.
0x2071	E_ENFS	Portmap failure on the host.
0x2072	E_ETIMEOUT	NFS call timed out.
0x2074	E_ENOAUTHBLK	No RPC authorization blocks are available.
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.
0x2077	E_PROBUNAVAIL	Program not available.
0x2078	E_EPROGVERSMISMATCH	Program version mismatched.
0x2079	E_ECANTDECODEARGS	Decode arguments error.
0x207A	E_EUNKNOWNHOST	Unknown host name.
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.
0x207C	E_UNKNOWNPROTO	Unknown protocol.
0x207D	E_EINTR	Call interrupted.
0x207E	ERPC	All other RPC errors.

Notes

1. If the specified filename already exists, the new file is not created. An existing file must first be deleted by using `remove_f()` before its name can be used for a new file.
2. `make_dir()` creates only directory files. `create_f()` creates an ordinary file.

See Also

`create_f`, `remove_f`, `open_dir`

mount_vol Mounts a pHILE+ formatted volume.

```
#include <phile.h>
unsigned long mount_vol(
    char *device,          /* volume name */
    unsigned long sync_mode /* synchronization mode */
)
```

Volume Types

pHILE+ formatted volumes.

Description

mount_vol() mounts a pHILE+ formatted volume. A volume must be mounted before any file operations can be applied to it. Permanent volumes (on non-removable media) need mounting only once. Removable volumes can be mounted and unmounted as required.

Arguments

device	Points to the null-terminated name of the volume to be mounted.
sync_mode	Specifies the volume's write synchronization attribute. The attribute is defined in <phile.h> and must be set to one of the following values.
SM_IMMED_WRITE	Immediate-write synchronization mode
SM_CONTROL_WRITE	Control-write synchronization mode
SM_DELAYED_WRITE	Delay-write synchronization mode
SM_READ_ONLY	Read-only synchronization mode

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2006	E_MNTHFULL	Attempted to mount too many volumes.
0x2007	E_VALIEN	Wrong volume format.
0x2008	E_MNTED	Volume already mounted.
0x2021	E_ILLLDEV	Illegal device (exceeded maximum.)
0x2024	E_FMODE	Illegal synchronization mode.
0x2025	E_IDN	Illegal device name.

2

Notes

1. `mount_vol()` proceeds as if the designated pSOS+ device were mountable. A device is mountable if it is a true storage device that has been initialized by `init_vol()`.
2. The number of volumes that can be mounted simultaneously in the system cannot exceed the pHILE+ Configuration Table parameter `fc_nmount`.
3. The pHILE+ file system manager operates without regard for volume ownership. Furthermore, any task can perform a `mount_vol()`, and a mounted device has no record of the task that mounted it. Therefore, a volume is not automatically unmounted when the task that mounted it is deleted. If these or any security measures need to be addressed, the user's own layer of software must do so.

See Also

`init_vol`, `pcmount_vol`, `nfsmount_vol`, `cdmount_vol`, `unmount_vol`

move_f

Moves (renames) a file.

```
#include <phile.h>
unsigned long move_f(
    char *oldname,    /* old pathname */
    char *newname     /* new pathname */
)
```

Volume Types

All volume types, except CD-ROM; however, some behavioral differences exist and are described here.

Description

`move_f()` changes the pathname associated with a file.

With one exception, the pHILE+ file system manager can move both ordinary and directory files on all volume types. The exception is directory files on MS-DOS formatted volumes, which cannot be moved. When a directory is moved, the directory and all files in the directory's subtree are moved.

Conceptually, `move_f()` moves a file by changing control structures on the volume (but no actual movement of data ever occurs). Therefore, `oldname` and `newname` must be on the same volume.

Arguments

`oldname` Points to the null-terminated old pathname.

`newname` Points to the null-terminated new pathname.

If `oldname` and `newname` are in the same directory, `move_f()` simply renames the file. Otherwise, `move_f()` has the effect of moving the file to a different location within the volume's directory tree. `move_f()` does not change the size or contents of the file.

`move_f()` fails if `newname` already exists or if the move operation would create a non-tree directory organization (for example, when a directory file is moved to its own subtree.)

If `oldname` is open, the file can be moved on pHILE+ and NFS volumes. An open file cannot be moved on MS-DOS volumes. Furthermore, no files can be moved on CD-ROM volumes.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2003	E_BADVOL	Inconsistent data on volume; volume corrupted.
0x200A	E_DMOUNT	Volume not mounted.
0x200B	E_FNAME	Filename not found.
0x200C	E_IFN	Illegal pathname.
0x200D	E_NDD	No default directory.
0x200E	E_FORD	Directory file expected.
0x2010	E_NODE	Null pathname.
0x2011	E_FEXIST	File already exists.
0x2012	E_FLIST	Too many files on volume.
0x2015	E_RO	Operation not allowed on read-only system files, directories, or mounted volumes.
0x2016	E_DIFDEV	Operation must be on the same volume.
0x2017	E_NOTREE	<code>move_f()</code> would destroy directory tree structure.
0x201C	E_ININFULL	Index block is full.
0x201D	E_VFULL	Volume is full.
0x2025	E_IDN	Illegal device name.
0x2051	E_MAXLOOP	Symbolic links are nested too deeply.
0x2052	E_EREMOTE	Too many levels of remote in path.

Hex	Mnemonic	Description
0x2054	E_EIO	A hard error happened at remote site.
0x2055	E_EACCES	Task does not have access permissions.
0x2056	E_EISDIR	Illegal operation on a directory.
0x2057	E_EQUOT	Quota exceeded.
0x2058	E_ESTALE	Stale NFS file handle.
0x205B	E_ENXIO	No such device or address.
0x205C	E_ENODEV	No such device.
0x2060	E_BADCD	CD-ROM volume; illegal operation.
0x2070	E_EAUTH	RPC authorization is not available.
0x2071	E_ENFS	Portmap failure on the host.
0x2072	E_ETIMEOUT	NFS call timed out.
0x2074	E_ENOAUTHBLK	No RPC authorization blocks are available.
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.
0x2077	E_PROBUNAVAIL	Program not available.
0x2078	E_EPROGVERSMISMATCH	Program version mismatched.
0x2079	E_ECANTDECODEARGS	Decode arguments error.
0x207A	E_EUNKNOWNHOST	Unknown host name.
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.
0x207C	E_UNKNOWNPROTO	Unknown protocol.
0x207D	E_EINTR	Call interrupted.
0x207E	ERPC	All other RPC errors.

See Also

make_dir

nfsmount_vol Mounts a remote file system.

```
#include <pna.h>      /* for htonl() */
#include <phile.h>
unsigned long nfsmount_vol(
    char *device,          /* volume name */
    NFSMOUNT_VOL_PARAMS *params /* parameters */
)
```

2

Volume Types

NFS volumes.

Description

`nfsmount_vol()` mounts an NFS volume. A volume must be mounted before any file operations can be conducted.

Arguments

device Points to a null-terminated name of the volume to be mounted. Unlike the `mount_vol()` system call, the volume name provided does not correspond to a true pSOS+ device but to a pseudo-device. A pseudo-device does not necessarily correspond to any real device or device driver in the pSOS+ system. Drivers for this device number may or may not exist. In either case, the pHILE+ file system manager does not call them while it is accessing the NFS volume.

params Points to an instance of the `nfsmount_vol_params` structure, which contains parameters used for volume mounting and is defined in `<phile.h>` as follows:

```
typedef struct nfsmount_vol_params {
    unsigned long ipaddr; /* Internet address of NFS server
                          * NOTE: network byte order */
    char *pathname;      /* pathname of filesystem to mount */
    unsigned long flags; /* reserved; set to 0 */
    unsigned long reserved[6]; /* reserved; set to 0 */
} NFSMOUNT_VOL_PARAMS;
```

This structure cannot be packed. The fields of `nfsmount_vol_params` are defined as follows:

<code>ipaddr</code>	The IP address of the NFS host that contains the file system to mount. Since this is in network byte order, it should be set as follows: <code>params->ipaddr = htonl(address)</code>
<code>pathname</code>	Points to the pathname of the filesystem to mount.
<code>flags</code>	Reserved for future use and must be 0.
<code>reserved</code>	Reserved for future use and must be 0.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2006	<code>E_MNTOFULL</code>	Attempted to mount too many volumes.
0x2008	<code>E_MNTOED</code>	Volume already mounted.
0x2025	<code>E_IDN</code>	Illegal device name.
0x2051	<code>E_MAXLOOP</code>	Symbolic links nested too deeply.
0x2052	<code>E_EREMOTE</code>	Too many levels of remote in path.
0x2054	<code>E_EIO</code>	A hard error occurred at a remote site.
0x2055	<code>E_EACCES</code>	Task does not have access permissions.
0x2058	<code>E_ESTALE</code>	Stale NFS file handle.
0x205B	<code>E_ENXIO</code>	No such device or address.
0x205C	<code>E_ENODEV</code>	No such device.
0x2070	<code>E_EAUTH</code>	RPC authorization is not available.
0x2071	<code>E_ENFS</code>	Portmap failure on the host.

Hex	Mnemonic	Description
0x2072	E_ETIMEDOUT	NFS call timed out.
0x2074	E_ENOAUTHBLK	No RPC authorization blocks are available.
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.
0x2077	E_PROBUNAVAIL	Program not available.
0x2078	E_EPROGVERSMISMATCH	Program version mismatched.
0x2079	E_ECANTDECODEARGS	Decode arguments error.
0x207A	E_EUNKNOWNHOST	Unknown host name.
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.
0x207C	E_UNKNOWNPROTO	Unknown protocol.
0x207D	E_EINTR	Call interrupted.
0x207E	ERPC	All other RPC errors.

Notes

1. The major device number of the volume name can exceed the maximum allowed device number in the pSOS+ Configuration Table because the device is *virtual*. A virtual device does not correspond to any device driver.
2. The number of volumes that can be mounted simultaneously in the system cannot exceed the pHILE+ Configuration Table parameter `fc_nmount`.

See Also

`mount_vol`, `pcmount_vol`, `cdmount_vol`, `unmount_vol`

open_dir Opens a directory file.

```
#include <phile.h>
unsigned long open_dir(
    char *dirname, /* name of the directory file */
    XDIR *dir /* pointer to buffer to return directory handle*/
)
```

Volume Types

All volume types.

Description

open_dir() opens a designated directory file.

Arguments

dirname	Points to a null-terminated pathname of a directory file.
dir	Points to an XDIR structure, which is defined in <phile.h>.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2003	E_BADVOL	Inconsistent data on volume; volume corrupted.
0x200C	E_IFN	Illegal pathname.
0x200D	E_NDD	No default directory.
0x2018	E_OFULL	Too many files open for task.
0x2019	E_NOFCB	Too many files open in system.
0x2025	E_IDN	Illegal device name.
0x2051	E_MAXLOOP	Symbolic links nested too deeply.

Hex	Mnemonic	Description
0x2052	E_EREMOTE	Too many levels of remote in path.
0x2054	E_EIO	A hard error occurred at a remote site.
0x2055	E_EACCES	Task does not have access permissions.
0x2058	E_ESTALE	Stale NFS file handle.
0x205B	E_ENXIO	No such device or address.
0x205C	E_ENODEV	No such device.
0x2070	E_EAUTH	RPC authorization is not available.
0x2071	E_ENFS	Portmap failure on the host.
0x2072	E_ETIMEOUT	NFS call timed out.
0x2074	E_ENOAUTHBLK	No RPC authorization blocks are available.
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.
0x2077	E_PROBUNAVAIL	Program not available.
0x2078	E_EPROGVERSMISMATCH	Program version mismatched.
0x2079	E_ECANTDECODEARGS	Decode arguments error.
0x207A	E_EUNKNOWNHOST	Unknown host name.
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.
0x207C	E_UNKNOWNPROTO	Unknown protocol.
0x207D	E_EINTR	Call interrupted.
0x207E	ERPC	All other RPC errors.

See Also

`close_dir`, `read_dir`

open_f Opens a file.

```
#include <phile.h>
unsigned long open_f(
    unsigned long *fid, /* file identifier */
    char *name,         /* pathname */
    unsigned long mode  /* unused; set to zero */
)
```

Volume Types

All volume types.

Description

`open_f()` creates a connection between a file and the calling task and returns a file identifier. The file identifier is used in subsequent operations on the file. `open_f()` fails if the system is out of file control blocks or if the task is out of open file table entries.

`open_f()` does not check for a file type. It opens ordinary files, directory files, or system files. However, directory files and system files are read-only.

`open_f()` always positions the `L_ptr` at the first byte in the file.

For CD-ROM volumes, `open_f()` can be used to read the *primary volume descriptor*. See “Primary Volume Descriptor,” under “Notes.”

Arguments

<code>fid</code>	Points to the variable where <code>open_f()</code> stores the file identifier.
<code>name</code>	Points to the null-terminated pathname of the file to open.
<code>mode</code>	Reserved for future use; should be set to 0 for future compatibility.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2003	E_BADVOL	Inconsistent data on volume; volume corrupted.
0x200A	E_DMOUNT	Volume not mounted.
0x200B	E_FNAME	Filename not found.
0x200C	E_IFN	Illegal pathname.
0x200D	E_NDD	No default directory.
0x200E	E_FORD	Directory file expected.
0x2018	E_OFULL	Too many files open for task.
0x2019	E_NOFCB	Too many files open in system.
0x2025	E_IDN	Illegal device name.
0x2051	E_MAXLOOP	Symbolic links nested too deeply.
0x2052	E_EREMOTE	Too many levels of remote in path.
0x2054	E_EIO	A hard error happened at remote site.
0x2055	E_EACCES	Task does not have access permissions.
0x2058	E_ESTALE	Stale NFS file handle.
0x205B	E_ENXIO	No such device or address.
0x205C	E_ENODEV	No such device.
0x2070	E_EAUTH	RPC authorization is not available.
0x2071	E_ENFS	Portmap failure on the host.
0x2072	E_ETIMEOUT	NFS call timed out.
0x2074	E_ENOAUTHBLK	No RPC authorization blocks are available.
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.
0x2077	E_PROBUNAVAIL	Program not available.
0x2078	E_EPROGVERSMISMATCH	Program version mismatched.

Hex	Mnemonic	Description
0x2079	E_ECANTDECODEARGS	Decode arguments error.
0x207A	E_EUNKNOWNHOST	Unknown host name.
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.
0x207C	E_EUNKNOWNPROTO	Unknown protocol.
0x207D	E_EINTR	Call interrupted.
0x207E	ERPC	All other RPC errors.

Notes

Primary Volume Descriptor

As a special case on CD-ROM volumes, the filename `_VOLUME.Y` in the root directory is used to read the primary volume descriptor, which is the starting point for locating all information on the volume. When you read `_VOLUME.Y`, pHILE+ omits the fields from it that are unused by your processor and byte-swaps the remaining fields to the proper order for the processor. Therefore, the primary volume descriptor can be read into the structure type that follows.


```

/* CD-ROM Primary Volume Descriptor as read from _VOLUME.$Y$ */
/*****

#define CDFS_NAMMAX    32    /* max node size (in bytes) */

/* CD File System Directory Record (internal format) */

typedef struct dir_cdfs {
    USHORT dr_reclen;           /* directory record length */
    USHORT dr_xarlen;          /* extended attribute record length */
    ULONG dr_extent;           /* number of first data block in file */
    ULONG dr_fsize;            /* byte size of file data space */
    ULONG dr_cdate;            /* date when created (pSOS+ format) */
    ULONG dr_ctime;            /* time when created (pSOS+ format) */
    USHORT dr_flags;           /* directory flags per iso_dirrec */
    USHORT dr_namlen;          /* byte length of name */
    char dr_name[CDFS_NAMMAX + 1] /*the name */
} dir_cdfs_t;

/* CD File System Volume Descriptor template returned to user upon read of */
/* "_VOLUME.$Y$" virtual file */

typedef struct desc_cdfs {
    UCHAR cd_type               /* volume descriptor type */
    UCHAR cd_id[5+1];          /* standard identifier */
    UCHAR cd_vers;             /* volume descriptor version */
    UCHAR cd_flags;            /* volume flags */
    UCHAR cd_sysid[32+1];      /* system identifier */
    UCHAR cd_volid[32+1];      /* volume identifier */
    ULONG cd_volsize;          /* volume space size */
    UCHAR cd_escseq[32];       /* escape sequences */
    USHORT cd_volsetsize;      /* volume set size */
    USHORT cd_volseqnum;       /* volume sequence number */
    USHORT cd_logblksiz;       /* logical block size */
    ULONG cd_pathtabsize;      /* path table byte size */
    ULONG cd_pathtab;          /* path table logical block */
    ULONG cd_pathtabopt;       /* opt path table log block */
    struct dir_cdfs cd_rootdir; /* root directory */
    UCHAR cd_volsetid[128+1];   /* volume set identifier */
    UCHAR cd_pubid[128+1];      /* publisher identifier */
    UCHAR cd_prepid[128+1];     /* data preparer identifier */
    UCHAR cd_appid[128+1];      /* application identifier */
    UCHAR cd_cpyrid[37+1];      /* copyright file identifier */
    UCHAR cd_absfid[37+1];      /* abstract file identifier */
    UCHAR cd_bibfid[37+1];      /* bibliographic identifier */
    ULONG cd_cdate;            /* volume create date (pSOS+ format) */
    ULONG cd_ctime;            /* volume create time (pSOS+ format) */
    ULONG cd_mdate;            /* modification time (pSOS+ format) */
    ULONG cd_xdate;            /* expiration date (pSOS+ format) */
    ULONG cd_xtime;            /* expiration time (pSOS+ format) */
    ULONG cd_edate;            /* effective date (pSOS+ format) */
    UCHAR cd_svers;            /* file structure version */
    UCHAR cd_appdata[512];      /* application private */
} desc_cdfs_t;

```

See Also

`open_fn`, `close_f`

open_fn Opens a file by its file identifier.

```
#include <phile.h>
unsigned long open_fn(
    unsigned long *fid,    /* file identifier */
    char *device,         /* volume name */
    unsigned long fn,     /* file number */
    unsigned long mode     /* unused, set to 0 */
)
```

2

Volume Types

pHILE+, MS-DOS, and CD-ROM formatted volumes.

Description

`open_fn()` functions identically to `open_f()` except that `open_fn()` opens a file associated with a specified file number. The file number is first obtained with `get_fn()`.

`open_fn()` is more efficient than `open_f()` when a particular file is frequently opened, since `open_fn()` skips pathname parsing and directory searching.

Arguments

<code>fid</code>	Points to the variable where <code>open_fn()</code> stores the file identifier.
<code>device</code>	Points to the null-terminated name of the volume containing the file.
<code>fn</code>	The file number of the file.
<code>mode</code>	Reserved for future use; should be set to 0 for future compatibility.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2003	E_BADVOL	Inconsistent data on volume; volume corrupted.
0x200A	E_DMOUNT	Volume not mounted.
0x200C	E_IFN	Illegal pathname.
0x200D	E_NDD	No default directory.
0x2018	E_OFULL	Too many files open for task.
0x2019	E_NOFCB	Too many files open in system.
0x2023	E_BADFN	Illegal or unused filename.
0x2025	E_IDN	Illegal device name.
0x2050	E_BADNFS	NFS volume; illegal operation.

Notes

Primary Volume Descriptor

As a special case on CD-ROM volumes, the filename `_VOLUME.Y` in the root directory is used to read the primary volume descriptor. Refer to the description of `open_f()` on page 2-78 for details.

See Also

`open_f`, `get_fn`, `close_f`

pcinit_vol Initializes an MS-DOS volume.

```
#include <phile.h>
unsigned long pcinit_vol(
    char *device,          /* volume name */
    void *scratch_buf,     /* scratch buffer */
    unsigned long dktype /* type of volume */
)
```

Volume Types

MS-DOS formatted volumes.

Description

pcinit_vol() initializes a volume in MS-DOS format. pcinit_vol() performs a logical format of the volume, setting up the necessary control structures and other information needed by the pHILE+ file system manager for subsequent file operations on the volume. A volume must be initialized before it can be mounted.

After a volume has been initialized, pcinit_vol() can be used to quickly delete all data on the volume.

pcinit_vol() cannot be used for the first initialization of a hard disk partition (see Note 4).

Arguments

device	Points to the null-terminated name of the volume to initialize.		
scratch_buf	Points to a 512-byte working buffer.		
dktype	Specifies the MS-DOS media format and must have one of the following values:		
	Value	Mnemonic	Meaning
	0	DK_HARD	Hard disk
	1	DK_360	360 Kbyte (5-1/4" double density)
	2	DK_12	1.2 Mbyte (5-1/4" high density)

3	DK_720	720 Kbyte (3-1/2" double density)
4	DK_144	1.44 Mbyte (3-1/2" high density)
5	DK_288	2.88 Mbyte (3-1/2" high density)
6	DK_NEC	1.2 Mbyte (5-1/4" NEC)
7	DK_OPT	Optical disks, 124.4 Mbyte (Fuji M2511A OMEM)

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2003	E_BADVOL	Inconsistent data on volume; volume corrupted.
0x2008	E_MNTED	Volume already mounted.
0x2025	E_IDN	Illegal device name.
0x2027	E_ILLMSTYP	Illegal DOS disk type.
0x2051	E_MAXLOOP	Symbolic links nested too deeply.

Notes

1. All data stored on the volume is lost by execution of this call.
2. The volume's hardware media must have been formatted before this call is executed.
3. A mounted volume cannot be initialized.
4. An MS-DOS volume must be formatted using the MS-DOS `FORMAT` utility or another comparable utility provided by some SCSI Controller Board vendors, not `pcinit_vol()`. After that, `pcinit_vol()` can be used to reinitialize the volume. `pcinit_vol()` reads the partition's boot record, not the master boot record (which was written by `FORMAT`), to get the partition's parameters.

See Also

`pcmount_vol`, `init_vol`

pcmount_vol Mounts an MS-DOS volume.

```
#include <phile.h>
unsigned long pcmount_vol(
    char *device,           /* volume name */
    unsigned long sync_mode /* synchronization mode */
)
```

Volume Types

MS-DOS formatted volumes.

Description

`pcmount_vol()` mounts an MS-DOS volume. A volume must be mounted before file operations can be applied to it.

Permanent (non-removable media) volumes need mounting only once. Removable volumes can be mounted and unmounted as required.

Arguments

<code>device</code>	Points to the null-terminated name of the volume to be mounted.								
<code>sync_mode</code>	Specifies the volume's write synchronization attribute. This attribute is defined in <code><phile.h></code> and must be set to one of the following values: <table> <tr> <td><code>SM_IMMED_WRITE</code></td><td>Immediate-write synchronization mode</td></tr> <tr> <td><code>SM_CONTROL_WRITE</code></td><td>Control-write synchronization mode</td></tr> <tr> <td><code>SM_DELAYED_WRITE</code></td><td>Delay-write synchronization mode</td></tr> <tr> <td><code>SM_READ_ONLY</code></td><td>Read-only synchronization mode</td></tr> </table>	<code>SM_IMMED_WRITE</code>	Immediate-write synchronization mode	<code>SM_CONTROL_WRITE</code>	Control-write synchronization mode	<code>SM_DELAYED_WRITE</code>	Delay-write synchronization mode	<code>SM_READ_ONLY</code>	Read-only synchronization mode
<code>SM_IMMED_WRITE</code>	Immediate-write synchronization mode								
<code>SM_CONTROL_WRITE</code>	Control-write synchronization mode								
<code>SM_DELAYED_WRITE</code>	Delay-write synchronization mode								
<code>SM_READ_ONLY</code>	Read-only synchronization mode								

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2003	E_BADVOL	Inconsistent data on volume; volume corrupted.
0x2006	E_MNTFULL	Attempted to mount too many volumes.
0x2007	E_VALIEN	Wrong volume format.
0x2008	E_MNTED	Volume is already mounted.
0x2021	E_ILLDEV	Illegal device (exceeded maximum).
0x2024	E_FMODE	Illegal synchronization mode.
0x2025	E_IDN	Illegal device name.
0x2029	E_NMSVOL	Cannot mount MS-DOS volume.
0x2051	E_MAXLOOP	Symbolic links nested too deeply.

2

Notes

1. `pcmount_vol()` proceeds as if the designated pSOS+ device is mountable. A device is mountable if it has been initialized by `pcinit_vol()` or by the MS-DOS `FORMAT` command.
2. The number of volumes that can be mounted simultaneously in the system cannot exceed the pHILE+ Configuration Table parameter `fc_nmount` (from `sys_conf.h`).
3. The pHILE+ file system manager does not attempt verification or any other way of determining volume ownership. Any task can perform a `pcmount_vol()`. A mounted device does not retain a record of the task that mounted it. Therefore, a volume is not automatically unmounted when the task that mounted it is deleted. This and any other security measures, if desired, should be supported by the user's own layer of software.
4. For an application to mount an MS-DOS volume, the mount flag `FC_MSDOS` in `sys_conf.h` must be set.

See Also

`pcinit_vol`, `mount_vol`, `nfsmount_vol`, `cdmount_vol`, `unmount_vol`

read_dir Reads directory entries in a file system independent format.

```
#include <phile.h>
unsigned long read_dir(
    XDIR *dir,          /* a directory handle */
    struct dirent *buf /* user structure to hold returned contents */
)
```

Volume Types

All volume types.

Description

read_dir() reads one directory entry at a time from a directory file in a file system-independent format. The directory handle is first obtained with open_dir().

Arguments

dir Points to the handle for the directory file, which has been returned by open_dir().

buf Points to the memory area that receives the data. The data returned in *buf is a dirent structure defined in <phile.h>, as follows:

```
struct dirent {
    unsigned long d_filno;
    char d_name [MAXNAMLEN+1];
}
```

This structure cannot be packed. d_fileno contains a number that is unique for each distinct file in the file system, and d_name contains a null-terminated filename, where the size is in the range of 1 through MAXNAMLEN+1. MAXNAMLEN is set to 255.

When the last entry has been read, an end-of-file error is returned.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2003	E_BADVOL	Inconsistent data on volume; volume corrupted.
0x201F	E_EOF	Read past end-of-file.
0x2052	E_EREMOTE	Too many levels of remote in path.
0x2054	E_EIO	A hard error occurred at a remote site.
0x2055	E_EACCES	Task does not have access permissions.
0x2058	E_ESTALE	Stale NFS file handle.
0x205A	E_NAMETOOLONG	Directory/filename too long.
0x205B	E_ENXIO	No such device or address.
0x205C	E_ENODEV	No such device.
0x2070	E_EAUTH	RPC authorization is not available.
0x2071	E_ENFS	Portmap failure on the host.
0x2072	E_ETIMEOUT	NFS call timed out.
0x2074	E_ENOAUTHBLK	No RPC authorization blocks are available.
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.
0x2077	E_PROBUNAVAIL	Program not available.
0x2078	E_EPROGVERSMISMATCH	Program version mismatched.
0x2079	E_ECANTDECODEARGS	Decode arguments error.
0x207A	E_EUNKNOWNHOST	Unknown host name.
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.
0x207C	E_UNKNOWNPROTO	Unknown protocol.
0x207D	E_EINTR	Call interrupted.
0x207E	ERPC	All other RPC errors.

Notes

Primary Volume Descriptor

As a special case on CD-ROM volumes, the filename `_VOLUME.Y` in the root directory is used to read the primary volume descriptor. Therefore, `_VOLUME.Y` is returned as one of the entries of the root directory. Refer to the description of `open_f()` on page 2-78 for details.

See Also

`open_dir`, `close_dir`

read_f Reads from a file.

```
#include <phile.h>
unsigned long read_f(
    unsigned long fid,      /* file identifier */
    void *buffer,          /* input buffer */
    unsigned long bcount,   /* byte read count */
    unsigned long *tcount   /* read count status */
)
```

2

Volume Types

All volume types.

Description

`read_f()` reads data from a file, beginning at the current position of the connection's `L_ptr`.

After `read_f()`, the file's `L_ptr` is updated to point to the byte after the last byte that was read.

Arguments

<code>fid</code>	Specifies the file identifier associated with the file.
<code>buffer</code>	Points to the memory area to receive the data.
<code>bcount</code>	Specifies the number of bytes to read.
<code>tcount</code>	Points to the variable where <code>read_f()</code> stores the number of bytes actually read. The <code>tcount</code> value equals <code>bcount</code> unless the end-of-file was reached or an error occurred.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2003	E_BADVOL	Inconsistent data on volume; volume corrupted.
0x201A	E_FIDBIG	Invalid FID; exceeds maximum.
0x201B	E_FIDOFF	Invalid FID; file closed.
0x2022	E_LOCKED	Data locked.
0x2052	E_EREMOTE	Too many levels of remote in path.
0x2054	E_EIO	A hard error occurred at remote site.
0x2055	E_EACCESS	Task does not have the necessary access permissions.
0x2056	E_EISDIR	Illegal operation on a directory.
0x2058	E_ESTALE	Stale NFS file handle.
0x205B	E_ENXIO	No such device or address.
0x205C	E_ENODEV	No such device.
0x2070	E_EAUTH	RPC authorization is not available.
0x2071	E_ENFS	Portmap failure on the host.
0x2072	E_ETIMEOUT	NFS call timed out.
0x2074	E_ENOAUTHBLK	No RPC authorization blocks are available.
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.
0x2077	E_PROBUNAVAIL	Program not available.
0x2078	E_EPROGVERSISMATCH	Program version mismatched.
0x2079	E_ECANTDECODEARGS	Decode arguments error.
0x207A	E_EUNKNOWNHOST	Unknown host name.
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.
0x207C	E_UNKNOWNPROTO	Unknown protocol.
0x207D	E_EINTR	Call interrupted.

Hex	Mnemonic	Description
0x207E	ERPC	All other RPC errors.

Notes

1. On pHILE+, CD-ROM, and MS-DOS formatted volumes, `read_f()` operations are more efficient if `bcount` equals an integral multiple of the block size and the `L_ptr` is positioned at a block boundary.
2. On pHILE+, CD-ROM, and MS-DOS formatted volumes, if the requested data includes entire blocks or a contiguous sequence of blocks and if such blocks are not already in the buffer cache, the pHILE+ file system manager reads these blocks directly into the caller's buffer (without going through the buffer cache).
3. `read_f()` automatically positions the `L_ptr` for sequential read operations. If random reads are necessary, use `lseek_f()` to reposition the `L_ptr`.

See Also

`lseek_f`, `read_vol`

read_link Reads the value of a symbolic link.

```
unsigned long read_link(  
    char *name, /* a file containing the symbolic link */  
    char *buf, /* user buffer to hold the returned contents */  
    unsigned long *bufsize /* maximum buffer size */  
)
```

Volume Types

NFS volumes.

Description

`read_link()` reads the contents of the symbolic link of a file. The returned data is *not null-terminated*.

Arguments

<code>name</code>	Points to the null-terminated pathname of the file containing the symbolic link.
<code>buf</code>	Points to the memory area that receives the data.
<code>bufsize</code>	Points to the maximum buffer size before the call, and the length of the data returned in <code>buf</code> after the call.

If successful, `read_link` stores in `*bufsize` the length of the data stored in `buf`. If this is the same as the maximum buffer size, only part of the data may have been returned.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2001	E_FUNC	pHILE+ format volume; illegal operation.
0x200C	E_IFN	Illegal pathname.
0x200D	E_NDD	No default directory.
0x2025	E_IDN	Illegal device name.
0x2026	E_BADMS	MS-DOS volume; illegal operation.
0x2051	E_MAXLOOP	Symbolic links nested too deeply.
0x2052	E_EREMOTE	Too many levels of remote in path.
0x2054	E_EIO	A hard error occurred at a remote site.
0x2055	E_EACCES	Task does not have access permissions.
0x2058	E_ESTALE	Stale NFS file handle.
0x205B	E_ENXIO	No such device or address.
0x205C	E_ENODEV	No such device.
0x2060	E_BADCD	CD-ROM volume; illegal operation.
0x2070	E_EAUTH	RPC authorization is not available.
0x2071	E_ENFS	Portmap failure on the host.
0x2072	E_ETIMEOUT	NFS call timed out.
0x2074	E_ENOAUTHBLK	No RPC authorization blocks are available.
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.
0x2077	E_PROBUNAVAIL	Program not available.
0x2078	E_EPROGVERSMISMATCH	Program version mismatched.
0x2079	E_ECANTDECODEARGS	Decode arguments error.
0x207A	E_EUNKNOWNHOST	Unknown host name.
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.

Hex	Mnemonic	Description
0x207C	E_UNKNOWNPROTO	Unknown protocol.
0x207D	E_EINTR	Call interrupted.
0x207E	ERPC	All other RPC errors.

Usage

The following example is a typical call to `read_link()` with all the code necessary for full error checking.

```
#define MAX_RESULT 100          /* Use 1 more than the longest
                                * expected result. */
{
    char contents[MAX_RESULT+1]; /* Contents of symbolic link */
    unsigned long size;          /* Size of contents */

    size = MAX_RESULT;          /* Room available in contents */

    if (read_link("3.2/sym_link", &contents[0], size) != 0)
        /* Error processing for failed system call */;

    contents[size] = '\0';       /* Null terminate the result */

    if (size == MAX_RESULT)      /* Possible partial result */
        /* Error processing for possible partial result */;
}
```

See Also

`lstat_f`, `symlink_f`

read_vol Reads directly from a pHILE+ formatted volume.

```
#include <phile.h>
unsigned long read_vol(
    char *device,          /* volume name */
    unsigned long block,   /* base block */
    unsigned long index,   /* byte offset */
    unsigned long bcount,  /* number of bytes to read */
    void *buffer           /* input buffer */
)
```

2

Volume Types

pHILE+, MS-DOS, and CD-ROM formatted volumes.

Description

`read_vol()` reads data directly from a volume, bypassing the file system organization imposed by the pHILE+ file system manager.

Arguments

<code>device</code>	Points to the null-terminated name of the volume to read.
<code>block</code>	Identifies the logical block number to begin reading.
<code>index</code>	Specifies where to begin reading within the specified block.
<code>bcount</code>	Specifies the number of bytes to read.
<code>buffer</code>	Points to the memory area to receive the data.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x200A	E_DMOUNT	Volume not mounted.
0x2025	E_IDN	Illegal device name.
0x2050	E_BADNFS	NFS volume; illegal operation.

Notes

1. If `index` is larger than the volume's block size, the read begins in a subsequent block. For example, on a volume with a 1024-byte block size, a read of block 5, index 1224, is the same as a read block 6, index 200.
2. CD-ROM volumes generally use a 2K block size.
3. `read_vol()` does not check for the end of the volume, so blocks beyond the specified volume size can be read if they physically exist.
4. If the requested data includes either entire blocks or a contiguous sequence of blocks and if such blocks are not already in the buffer cache, the pHILE+ file system manager reads blocks directly into the buffer (without going through the buffer cache). Therefore, `read_vol()` executes more efficiently when `bcount` and `index` are equal to integral multiples of blocks.

See Also

`write_vol`

remove_f Deletes a file.

```
#include <phile.h>
unsigned long remove_f(
    char *name    /* pathname */
)
```

Volume Types

All volume types, except CD-ROM; however, functional differences exist and are described here.

Description

remove_f() deletes a file from a volume. The file can be an ordinary file or a directory file. All storage used by the file is returned to the system for reuse. The file's entry in its parent directory is also deleted.

System files and non-empty directory files cannot be deleted. On pHILE+ and MS-DOS formatted volumes, an open file cannot be deleted. An open file can be deleted on an NFS volume. CD-ROM volumes are read-only.

Arguments

name Points to the null-terminated pathname of the file to delete.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2003	E_BADVOL	Inconsistent data on volume; volume corrupted.
0x200A	E_DMOUNT	Volume not mounted.
0x200B	E_FNAME	Filename not found.

Hex	Mnemonic	Description
0x200C	E_IFN	Illegal pathname.
0x200D	E_NDD	No default directory.
0x200E	E_FORD	Directory file expected.
0x2010	E_NODE	Null pathname.
0x2013	E_FOPEN	Cannot remove an open file.
0x2014	E_DNE	Directory not empty.
0x2015	E_RO	Operation not allowed on read-only system files, directories, or mounted volumes.
0x2025	E_IDN	Illegal device name.
0x2051	E_MAXLOOP	Symbolic links nested too deeply.
0x2052	E_EREMOTE	Too many levels of remote in path.
0x2053	E_PERM	Task does not have ownership.
0x2054	E_EIO	A hard error occurred at remote site.
0x2055	E_EACCES	Task does not have access permissions.
0x2056	E_EISDIR	Illegal operation on a directory.
0x2058	E_ESTALE	Stale NFS file handle.
0x205B	E_ENXIO	No such device or address.
0x205C	E_ENODEV	No such device.
0x2060	E_BADCD	CD-ROM volume; illegal operation.
0x2070	E_EAUTH	RPC authorization is not available.
0x2071	E_ENFS	Portmap failure on the host.
0x2072	E_ETIMEDOUT	NFS call timed out.
0x2074	E_ENOAUTHBLK	No RPC authorization blocks are available.
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.
0x2077	E_PROBUNAVAIL	Program not available.

Hex	Mnemonic	Description
0x2078	E_EPROGVERSMISMATCH	Program version mismatched.
0x2079	E_ECANTDECODEARGS	Decode arguments error.
0x207A	E_EUNKNOWNHOST	Unknown host name.
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.
0x207C	E_UNKNOWNPROTO	Unknown protocol.
0x207D	E_EINTR	Call interrupted.
0x207E	ERPC	All other RPC errors.

See Also

create_f, make_dir

stat_f Gets the status of a named file.

```
#include <phile.h>
unsigned long stat_f(
    char *name,          /* file pathname */
    struct stat *buf      /* file status */
)
```

Volume Types

All volumes.

Description

stat_f() returns information about the named file. This call does not need read, write, or execute permission of the named file. It does need execute/search permission of all the directories leading to the named file.

Arguments

name	Points to the null-terminated pathname of the file.
buf	Points to a stat structure defined in <phile.h> as follows:

```
struct stat {
    mode_t st_mode;      /* ownership/protection */
    ino_t st_ino;        /* file ID */
    dev_t st_dev;        /* device ID where the volume resides */
    dev_t st_rdev;       /* device ID, for character or block
                        * special files only */
    nlink_t st_nlink;    /* number of hard links to the file */
    uid_t st_uid;        /* user ID */
    gid_t st_gid;        /* group ID */
    off_t st_size;       /* total size of file, in bytes */
    time_t st_atime;     /* file last access time */
    time_t st_mtime;     /* file last modify time */
    time_t st_ctime;     /* file last status change time */
    long st_blksize;     /* optimal block size for I/O ops */
    long st_blocks;      /* file size in blocks */
};
```


This structure cannot be packed. mode_t, ino_t, dev_t, nlink_t, uid_t, gid_t, off_t, and time_t are defined as unsigned long in <phile.h>. The following differences exist for local file systems (pHILE+, MS-DOS, and CD-ROM):

```
rdev = dev, nlink = 1, uid = 0, gid = 0, atime = ctime
= mtime
```

The status information word st_mode contains the following bits:

_IFMT	0170000	/* type of file */
_IFIFO	0010000	/* fifo special */
_IFCHR	0020000	/* character special */
_IFDIR	0040000	/* directory */
_IFBLK	0060000	/* block special */
_IFREG	0100000	/* regular file */
_IFLNK	0120000	/* symbolic link */
_IFSOCK	0140000	/* socket */
S_ISUID	0004000	/* set user ID on execution */
S_ISGID	0002000	/* set group ID on execution */
S_ISVTX	0001000	/* save swapped text even after use */
S_IRUSR	0000400	/* read permission, owner */
S_IWUSR	0000200	/* write permission, owner */
S_IXUSR	0000100	/* execute/search permission, owner */
S_IRGRP	0000040	/* read permission, group */
S_IWGRP	0000020	/* write permission, group */
S_IXGRP	0000010	/* execute/search permission, group */
S_IROTH	0000004	/* read permission, other */
S_IWOTH	0000002	/* write permission, other */
S_IXOTH	0000001	/* execute/search permission, other */

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2003	E_BADVOL	Inconsistent data on volume; volume corrupted.
0x200C	E_IFN	Illegal pathname.
0x200D	E_NDD	No default directory.
0x2025	E_IDN	Illegal device name.
0x2051	E_MAXLOOP	Symbolic links nested too deeply.
0x2052	E_EREMOTE	Too many levels of remote in path.
0x2054	E_EIO	A hard error occurred at a remote site.
0x2055	E_EACCES	Task does not have access permissions.
0x2058	E_ESTALE	Stale NFS file handle.
0x205B	E_ENXIO	No such device or address.
0x205C	E_ENODEV	No such device.
0x2070	E_EAUTH	RPC authorization is not available.
0x2071	E_ENFS	Portmap failure on the host.
0x2072	E_ETIMEDOUT	NFS call timed out.
0x2074	E_ENOAUTHBLK	No RPC authorization blocks are available.
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.
0x2077	E_PROBUNAVAIL	Program not available.
0x2078	E_EPROGVERSMISMATCH	Program version mismatched.
0x2079	E_ECANTDECODEARGS	Decode arguments error.
0x207A	E_EUNKNOWNHOST	Unknown host name.
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.
0x207C	E_UNKNOWNPROTO	Unknown protocol.
0x207D	E_EINTR	Call interrupted.

Hex	Mnemonic	Description
0x207E	ERPC	All other RPC errors.

Usage

`stat_f()` can be used to determine both the current device and the current directory of local volumes. Thus you can use `stat_f()` to return to a device and directory after leaving them, or to construct absolute path names starting at the current directory. Only the directory file number is available, not the full directory path. To obtain the full directory path, see `get_fn()`.

```
/* Obtaining both the current device and the current directory */

ULONG rc;                                /* System call return code */
struct stat  current_stat;               /* stat_f() of "." */
ULONG device;                            /* Current device */
ULONG directory;                         /* Current directory */

if((rc = stat_f(".", &current_stat)) != 0)
    /* Error processing */

device = current_stat.st_dev;
directory = current_stat.st_dev;

/* Returning to the above device and directory at a later time. */

char directory[29];                      /* To change back */

sprintf(directory, "0x%04x.0x%02x.0x%02x.0x%08x/.",
    device >> 16,                        /* Major device number */
    (device >> 8) & 0xffU,               /* Minor device number */
    device & 0xffU,                      /* Partition number */
    directory);                          /* File number to start at */

if((rc = change_dir(directory)) != 0)
    /* Error processing */

/* Constructing absolute path name starting at the saved directory */

#define REL_PATH_LEN 8                  /* Length of path below saved
                                         * directory */
char path[28 + PATH_LEN];               /* Absolute path of file.txt */

sprintf(path, "0x%04x.0x%02x.0x%02x.0x%08x/%s",
    device >> 16,                        /* Major device number */
    (device >> 8) & 0xffU,               /* Minor device number */
    device & 0xffU,                      /* Partition number */
    directory);
```

```
directory,      /* File number to start at */  
"file.txt"); /* Relative path below saved directory */
```

See Also

fstat_f, chmod_f, fchmod_f, chown_f, fchown_f, link_f, read_f,
read_link, truncate_f, ftruncate_f, remove_f, utime_f, write_f

stat_vfs Gets statistics for a named volume.

```
#include <phile.h>
unsigned long stat_vfs(
    char *name,           /* file pathname */
    struct statvfs *buf    /* volume statistics */
)
```

2

Volume Types

All volume types.

Description

`stat_vfs()` returns information about a mounted volume.

Arguments

name Points to a null-terminated pathname of any file within the mounted volume.

buf Points to a `statvfs` structure defined in `<phile.h>`, as follows:

```
typedef struct {
    long val[2];
} fsid_t;

struct statvfs {
    unsigned long f_bsize;      /* preferred volume block size */
    unsigned long f_frsize;    /* fundamental volume block size */
    unsigned long f_blocks;     /* total number of blocks */
    unsigned long f_bfree;     /* total number of free blocks */
    unsigned long f_bavail;    /* free blocks available to
                               * non-superuser */
    unsigned long f_files;     /* total # of file nodes
                               * (pHILE+ files only) */
    unsigned long f_ffree;     /* reserved (not supported) */
    unsigned long f_favail;    /* reserved (not supported) */
    fsid_t f_fsid;             /* reserved (not supported) */
    char f_basetype[16];       /* reserved (not supported) */
    unsigned long f_flag;      /* reserved (not supported) */
    unsigned long f_namemax;   /* reserved (not supported) */
    char f_fstr[32];           /* reserved (not supported) */
    unsigned long f_fstype;    /* file system type number */
    unsigned long f_filler[15];/* reserved (not supported) */
};
```

This structure cannot be packed. Currently, the fields `f_ffree`, `f_favail`, `f_fsid`, `f_basetype`, `f_flag`, `f_namemax`, `f_fstr` and `f_filler` are reserved and do not have values. For all volumes except pHILE+ format, the field `f_files` is unused.

The field `f_fstype` identifies the type of file system format. The values in `<phile.h>` are given below:

<code>FSTYPE_PHILE</code>	pHILE+ format volume
<code>FSTYPE_PCDOS</code>	MS-DOS format volume
<code>FSTYPE_CDROM</code>	CD-ROM format volume
<code>FSTYPE_NFS</code>	Client NFS volume

The return value for all unsupported fields is 0.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2003	<code>E_BADVOL</code>	Inconsistent data on volume; volume corrupted.
0x200C	<code>E_IFN</code>	Illegal pathname.
0x200D	<code>E_NDD</code>	No default directory.
0x2025	<code>E_IDN</code>	Illegal device name.
0x2051	<code>E_MAXLOOP</code>	Symbolic links nested too deeply.
0x2052	<code>E_EREMOTE</code>	Too many levels of remote in path.
0x2054	<code>E_EIO</code>	A hard error occurred at a remote site.
0x2055	<code>E_EACCES</code>	Task does not have access permissions.
0x2058	<code>E_ESTALE</code>	Stale NFS file handle.
0x205B	<code>E_ENXIO</code>	No such device or address.
0x205C	<code>E_ENODEV</code>	No such device.

Hex	Mnemonic	Description
0x2070	E_EAUTH	RPC authorization is not available.
0x2071	E_ENFS	Portmap failure on the host.
0x2072	E_ETIMEOUT	NFS call timed out.
0x2074	E_ENOAUTHBLK	No RPC authorization blocks are available.
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.
0x2077	E_PROBUNAVAIL	Program not available.
0x2078	E_EPROGVERSMISMATCH	Program version mismatched.
0x2079	E_ECANTDECODEARGS	Decode arguments error.
0x207A	E_EUNKNOWNHOST	Unknown host name.
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.
0x207C	E_UNKNOWNPROTO	Unknown protocol.
0x207D	E_EINTR	Call interrupted.
0x207E	ERPC	All other RPC errors.

See Also

fstat_vfs

symlink_f Creates a symbolic link to a file.

```
unsigned long symlink_f(
    char *name1,    /* a string used in creating the
                    * symbolic link */
    char *name2     /* the name of the file to be created */
)
```

Volume Types

NFS volumes.

Description

`symlink_f()` creates a symbolic link `name1` in the file `name2`. The files do not need to be on the same volume.

The file to which the symbolic link points is used when an `open_f()` is performed on the link. A `stat_f()` performed on a symbolic link returns the linked-to file (whereas `lstat_f()` returns information about the link itself). `read_link()` can be used to read the contents of a symbolic link.

Arguments

<code>name1</code>	Points to the null-terminated pathname of the symbolic link.
<code>name2</code>	Points to the null-terminated pathname of the file.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2001	<code>E_FUNC</code>	Invalid function number.
0x200C	<code>E_IFN</code>	Illegal pathname.
0x200D	<code>E_NDD</code>	No default directory.

Hex	Mnemonic	Description
0x2025	E_IDN	Illegal device name.
0x2026	E_BADMS	MS-DOS volume; illegal operation.
0x2051	E_MAXLOOP	Symbolic links nested too deeply.
0x2052	E_EREMOTE	Too many levels of remote in path.
0x2054	E_EIO	A hard error occurred at a remote site.
0x2055	E_EACCES	Task does not have access permissions.
0x2057	E_EQUOT	Quota exceeded.
0x2058	E_ESTALE	Stale NFS file handle.
0x205B	E_ENXIO	No such device or address.
0x205C	E_ENODEV	No such device.
0x2060	E_BADCD	CD-ROM volume; illegal operation.
0x2070	E_EAUTH	RPC authorization is not available.
0x2071	E_ENFS	Portmap failure on the host.
0x2072	E_TIMEDOUT	NFS call timed out.
0x2074	E_ENOAUTHBLK	No RPC authorization blocks are available.
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.
0x2077	E_PROBUNAVAIL	Program not available.
0x2078	E_EPROGVERSMISMATCH	Program version mismatched.
0x2079	E_ECANTDECODEARGS	Decode arguments error.
0x207A	E_EUNKNOWNHOST	Unknown host name.
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.
0x207C	E_UNKNOWNPROTO	Unknown protocol.
0x207D	E_EINTR	Call interrupted.
0x207E	ERPC	All other RPC errors.

See Also

`read_link`, `link_f`, `remove_f`

sync_vol Synchronizes a volume.

```
#include <phile.h>
unsigned long sync_vol(
    char *device    /* volume name */
)
```

Volume Types

pHILE+ and MS-DOS formatted volumes.

Description

sync_vol() updates a mounted volume by writing all modified volume information to the physical device. Updated files, descriptors, and all cache buffers that contain physical blocks are flushed to the device.

This call enables manual updating of a volume and is irrelevant in relation to immediate-write synchronization mode. CD-ROM volumes are read-only.

Arguments

device Points to the null-terminated name of the volume to synchronize.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2003	E_BADVOL	Inconsistent data on volume; volume corrupted.
0x200A	E_DMOUNT	Volume not mounted.
0x2025	E_IDN	Illegal device name.
0x2050	E_BADNFS	NFS volume; illegal operation.

Hex	Mnemonic	Description
0x2060	E_BADCD	CD-ROM volume; illegal operation.

Notes

Because no inherent access restrictions exist with respect to a volume, any task can call `sync_vol()`. `sync_vol()` keeps the volume busy during the update.

See Also

`unmount_vol`, `mount_vol`, `pcmount_vol`

truncate_f Changes the size of a named file.

```
#include <phile.h>
unsigned long truncate_f(
    char *name,           /* file pathname */
    unsigned long length  /* file size in bytes */
)
```

2

Volume Types

pHILE+, MS-DOS, and NFS volumes.

Description

`truncate_f()` causes the file specified by `name` to have a size (in bytes) equal to `length`. If the file was previously longer than `length`, the extra bytes are truncated. If it was shorter, the bytes between the old and new lengths are filled with zeroes.

Unlike `annex_f()`, this system call changes both the logical and the physical file size. (`annex_f()` changes only the physical file size.)

On pHILE+ or MS-DOS volumes, the file must not be open. If this is violated, the error `E_FOPEN` is returned.

Arguments

<code>name</code>	Points to the null-terminated pathname of the file.
<code>length</code>	Specifies the new file size in bytes.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2001	E_FUNC	Invalid function number.
0x2003	E_BADVOL	Inconsistent data on volume; volume corrupted.
0x200C	E_IFN	Illegal pathname.
0x200D	E_NDD	No default directory.
0x2015	E_RO	Operation not allowed on read-only system files, directories, or mounted volumes.
0x201C	E_ININFULL	Index block full.
0x201D	E_VFULL	Volume is full. (This cannot happen on NFS volumes.)
0x2022	E_LOCKED	Data is locked.
0x2025	E_IDN	Illegal device name.
0x2051	E_MAXLOOP	Symbolic links nested too deeply.
0x2052	E_EREMOTE	Too many levels of remote in path.
0x2054	E_EIO	A hard error occurred at a remote site.
0x2055	E_EACCES	Task does not have access permissions.
0x2057	E_EQUOT	Quota exceeded.
0x2058	E_ESTALE	Stale NFS file handle.
0x205B	E_ENXIO	No such device or address.
0x205C	E_ENODEV	No such device.
0x2060	E_BADCD	CD-ROM volume; illegal operation.
0x2070	E_EAUTH	RPC authorization is not available.
0x2071	E_ENFS	Portmap failure on the host.
0x2072	E_ETIMEDOUT	NFS call timed out.
0x2074	E_ENOAUTHBLK	No RPC authorization blocks are available.

Hex	Mnemonic	Description
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.
0x2077	E_PROBUNAVAIL	Program not available.
0x2078	E_EPROGVERSMISMATCH	Program version mismatched.
0x2079	E_ECANTDECODEARGS	Decode arguments error.
0x207A	E_EUNKNOWNHOST	Unknown host name.
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.
0x207C	E_UNKNOWNPROTO	Unknown protocol.
0x207D	E_EINTR	Call interrupted.
0x207E	ERPC	All other RPC errors.

See Also

ftruncate_f, open_f, open_fn

unmount_vol Unmounts a volume.

```
#include <phile.h>
unsigned long unmount_vol(
    char *device    /* volume name */
)
```

Volume Types

All volume types.

Description

`unmount_vol()` unmounts a previously mounted volume. Unmounting a volume causes it to be synchronized. Synchronization causes all memory-resident volume data to be flushed to the device.

Arguments

`device` Points to the null-terminated name of the volume to unmount.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2003	E_BADVOL	Inconsistent data on volume; volume corrupted.
0x2009	E_MNTOOPEN	Files are open on volume.
0x200A	E_DMOUNT	Volume not mounted.
0x200C	E_IFN	Illegal pathname.
0x200D	E_NDD	No default directory.
0x2025	E_IDN	Illegal device name.

Hex	Mnemonic	Description
0x2051	E_MAXLOOP	Symbolic links nested too deeply.
0x2052	E_EREMOTE	Too many levels of remote in path.
0x2054	E_EIO	A hard error happened at remote site.
0x2058	E_ESTALE	Stale NFS file handle.
0x205B	E_ENXIO	No such device or address.
0x205C	E_ENODEV	No such device.
0x2070	E_EAUTH	RPC authorization is not available.
0x2071	E_ENFS	Portmap failure on the host.
0x2072	E_ETIMEOUT	NFS call timed out.
0x2074	E_ENOAUTHBLK	No RPC authorization blocks are available.
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.
0x2077	E_PROBUNAVAIL	Program not available.
0x2078	E_EPROGVERSMISMATCH	Program version mismatched.
0x2079	E_ECANTDECODEARGS	Decode arguments error.
0x207A	E_EUNKNOWNHOST	Unknown host name.
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.
0x207C	E_UNKNOWNPROTO	Unknown protocol.
0x207D	E_EINTR	Call interrupted.
0x207E	ERPC	All other RPC errors.

Notes

1. Any task can unmount a volume. If some security is needed, the user must supply the bookkeeping software to keep track of volumes and tasks that perform the mounts.
2. Conceptually, unmounting a volume is unnecessary unless it is physically removed and a new volume is mounted on the same device. However, a limit exists to the number of volumes that can be mounted simultaneously, and unmounting frees entries in the volume mount table.

3. `unmount_vol()` fails and returns an error if any open files exist on the volume.
4. Once unmounted, a volume is inaccessible.

See Also

`mount_vol`, `pcmount_vol`, `nfsmount_vol`, `cdmount_vol`

utime_f Sets the access and modification times of a file.

```
#include <phile.h>
unsigned long utime_f(
    char *name,          /* file pathname */
    struct utimbuf *times /* file access and modification times */
)
```

2

Volume Types

NFS volumes.

Description

`utime_f()` sets the access and modification times of a file.

Arguments

<code>name</code>	Points to the null-terminated pathname of the file.
<code>times</code>	If <code>times</code> is <code>NULL</code> , the access and modification times are set to the current time. Otherwise, <code>times</code> is interpreted as a pointer to a <code>utimbuf</code> structure defined in <code><phile.h></code> as follows:

```
struct utimbuf {
    time_t actime; /* access time */
    time_t modtime; /* modification time */
};
```

This structure cannot be packed. No time zone is associated with the time values.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2001	E_FUNC	Invalid function number.
0x200C	E_IFN	Illegal pathname.
0x200D	E_NDD	No default directory.
0x2015	E_RO	Operation not allowed on read-only system files, directories, or mounted volumes.
0x2025	E_IDN	Illegal device name
0x2026	E_BADMS	MS-DOS volume; illegal operation.
0x2051	E_MAXLOOP	Symbolic links nested too deeply.
0x2052	E_EREMOTE	Too many levels of remote in path.
0x2054	E_EIO	A hard error occurred at a remote site.
0x2055	E_EACCES	Task does not have access permissions.
0x2058	E_ESTALE	Stale NFS file handle.
0x205B	E_ENXIO	No such device or address.
0x205C	E_ENODEV	No such device.
0x2060	E_BADCD	CD-ROM volume; illegal operation.
0x2070	E_EAUTH	RPC authorization is not available.
0x2071	E_ENFS	Portmap failure on the host
0x2072	E_ETIMEDOUT	NFS call timed out.
0x2074	E_ENOAUTHBLK	No RPC authorization blocks are available.
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.
0x2077	E_PROBUNAVAIL	Program not available.
0x2078	E_PROGVERSMISMATCH	Program version mismatched.
0x2079	E_ECANTDECODEARGS	Decode arguments error.

Hex	Mnemonic	Description
0x207A	E_EUNKNOWNHOST	Unknown host name.
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.
0x207C	E_UNKNOWNPROTO	Unknown protocol.
0x207D	E_EINTR	Call interrupted.
0x207E	ERPC	All other RPC errors.

See Also

stat_f, fstat_f

verify_vol

Verifies a volume's control structures.

```
#include <phile.h>
unsigned long verify_vol(
    char *device,          /* volume name */
    VERIFY_VOL_PARAMS *params /* parameters */
)
```

Volume Types

pHILE+ formatted volumes.

Description

verify_vol() examines all control structures on a pHILE+ formatted volume. Inconsistencies are reported to a user-supplied callout routine, which can relay further instructions. The callout routine can then request verify_vol() to correct the inconsistency.

Usage instructions for verify_vol are provided in “Usage” on page 2-129.

Arguments

device	Points to the null-terminated name of the volume to be verified.
params	Points to an instance of the verify_vol_params structure defined in <phile.h> as follows:

```
typedef struct verify_vol_params {
    void *pb_dataptr;          /* work area pointer */
    unsigned long pb_datalen;  /* length of work area */
    unsigned long pb_maxdepth; /* maximum depth of
                               * directory tree */
    fault_desc_block *pb_fdbptr; /* fault descriptor block
                               * pointer */
    unsigned long (*pb_faultp)(void); /* faultp function */
    unsigned long *pb_badblkptr; /* bad block list */
} VERIFY_VOL_PARAMS;
```

This structure cannot be packed.

The contents of the `verify_vol_params` fields are as follows:

`pb_dataptr` Points to a work area required by `verify_vol()`. The size of this work area (in bytes) is given by the formula:

$$96 + (4 * vsize) + (16 * nfd) + (38 * maxdepth)$$

where *vsize* is the size of the volume in blocks; *nfd* is the number of file descriptors specified when the volume was initialized; and *maxdepth* is the maximum depth of the directory tree (it should be equal to `pb_maxdepth`, below.) For example, on a volume with:

vsize = 5000 blocks

nfd = 100 entries

maxdepth = 5 levels

a total of $96 + (4 * 5000) + (16 * 100) + (38 * 5) = 21886$ bytes would be required. This work area can be statically allocated, or it can be dynamically allocated by a `pSOS+ rn_getseg()` call.

`pb_dataalen` The size (in bytes) of the work area pointed to by `pb_dataptr`. The `verify_vol()` call uses this entry to confirm that the work area is large enough.

`pb_maxdepth` The maximum depth of the volume's directory tree. If any branch of the directory exceeds this depth, `verify_vol()` terminates and returns an error code to the calling task. The minimum value allowed is 1, which indicates a flat directory (i.e., one containing no subdirectories).

`pb_fdbptr` Points to a fault descriptor block (FDB) in the caller's memory area. When a fault is detected, `verify_vol()` places a detailed description of the fault into the FDB. The FDB format is described on page 2-132.

`pb_faultp` Points to the user-provided `faultp()` procedure that is called each time `verify_vol()` detects a fault. `faultp()` is responsible for processing the fault. Refer to "faultp()" on page 2-131 for more details.

`pb_badblkptr` Points to a user-provided list of *bad blocks* on the volume. A bad block is a block that cannot be read and/or written and is therefore unusable by the pHILE+ file system manager. This list is made up of 32-bit entries and is terminated with a 0 entry. The entries need not be in any specific order. `verify_vol()` can greatly simplify the handling of bad blocks. Refer to “Bad Blocks” on page 2-139 for information on this feature. If no bad block list is provided, this entry must be 0.

Target

`verify_vol()` calls a user-supplied function, `faultp()`, for status-checking (see page 2-131). For each processor family, `faultp()` returns its return value in the register specified below:



On 68K processors, `faultp()` uses the D0.L register.



On PowerPC processors, `faultp()` uses the r3 register.



On 960 processors, `faultp()` uses the g0 register.



On x86 processors, `faultp()` uses the %eax register.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2007	E_VALIEN	Wrong volume format.
0x2009	E_MNTOPEN	Files are open on volume.

Hex	Mnemonic	Description
0x200A	E_DMOUNT	Volume not mounted.
0x2021	E_ILLDEV	Illegal device (exceeded maximum).
0x2025	E_IDN	Illegal device name.
0x2051	E_MAXLOOP	Symbolic links nested too deeply.
0x2200	VF_INSUFF	Insufficient working area provided.
0x2201	VF_MAXDEPTH	Maximum depth exceeded on directory traversal.
0x2202	VF_ABORT	Verify routine aborted by user.

Usage

`verify_vol()` can be used to perform the following actions:

Volume Integrity Verification — `verify_vol()` examines all volume control structures to verify their consistency. Inconsistencies are reported and described in detail.

Volume Correction — Certain kinds of inconsistencies can be corrected.

Bad Block Elimination — Bad blocks can be marked as “in use” in the volume bitmap, thus excluding them from allocation by pHILE+ file system manager.

`verify_vol()` can be used in two ways. First, it can be used to perform a simple test of correctness, for example, at each power-on or system restart. Second, it can be integrated into a volume repair utility with a user-supplied interface.

Under normal operating conditions, pHILE+ file system manager always maintains the volume control structures in a correct and consistent state. `verify_vol()` is most useful when used following a system error or failure that can corrupt the file system, such as one of the following:

- A power failure, or a CPU or disk controller crash. In such cases, pHILE+ file system manager can be interrupted in the middle of a critical operation, resulting in a corrupted file system.
- A hard error or data corruption in one or more blocks containing volume control structures.
- Errors in the user-supplied physical disk driver.
- Restarting a task in pHILE+ file system manager.

Requirements and Restrictions

1. `verify_vol()` suspends all other I/O transactions to the designated volume. Because of the time required to execute it, `verify_vol()` should be called when the volume is idle.
2. Executing `verify_vol()` requires that the volume is mounted and no files are open.
3. `verify_vol()` cannot be used on an MS-DOS, CD-ROM, or NFS volume.

Functional Description

On the specified volume, `verify_vol()` examines the volume's control structures and searches for *faults*. A fault is any inconsistency in the control structures. For example, the volume's bitmap may indicate a particular block is free, while in fact it is being used. In all, there are 42 different kinds of faults detectable by `verify_vol()`.

`verify_vol()` examines the following volume control structures:

- Root block
- Bitmap
- FLIST
- All directories
- All file indirect blocks
- All file index blocks

`verify_vol()` stores a detailed description of the detected fault into a user-provided fault descriptor block (FDB) and then calls the user-provided function `faultp()`, described below.

faultp()

`faultp()` is called by `verify_vol()` without any parameters. `faultp()` is responsible for additional processing of the fault.

`verify_vol()` calls `faultp()` with the following information:

- The type of fault
- A detailed description of the fault
- An indication of whether or not the fault is correctable

`faultp()` performs its own check and returns a status code in the register supported by pHILE+, which is processor-specific. The register used on each processor family is specified in *Target* on page 2-128.

The status code `faultp()` returns must be one of the following:

Code	Description
0	Continue volume verification without correcting the fault.
1	Correct fault and continue volume verification.
2	Terminate volume verification.

If `status = 1` is returned, `verify_vol()` makes modifications to the volume, which correct the fault. In most cases, the obvious modification is made. Less obvious modifications are described in “Bad Blocks” on page 2-139. Note that `status = 1`

(correct fault) should be returned only if the FDB indicates the fault is fixable (refer to the FDB description below). If status = 1 is returned for a non-fixable fault, it is ignored and verification continues.

If `verify_vol()` is being used simply to verify volume correctness, then, when called, `faultp()` can return a status of 0, which continues the rest of volume verification, or a status of 2, which terminates `verify_vol()` and returns an error to the caller.

`verify_vol()` can also be integrated into a “volume repair” utility with an operator interface. This utility should implement `faultp()` so that it will

- Display each fault in detail;
- Indicate if the fault is fixable;
- And if so, ask the user if he wants it fixed;
- And if so, return status = 1 to `verify_vol()`.

Faults that are not fixable may require additional user action. For example, you may perform the following steps repeatedly:

1. Use `verify_vol()` to correct all fixable errors and obtain a list of non-fixable errors.
2. Examine, copy, and delete the affected files, as required.

When step 1 produces no more faults, you can consider the volume corrected.

The Fault Descriptor Block (FDB)

The structure `fault_desc_block` defines the FDB in `phile.h` as follows:

```
typedef struct fault_desc_block {
    unsigned long fdb_code;    /* fault code */
    unsigned long fdb_fn1;    /* file number for file 1 */
    unsigned long fdb_fn2;    /* file number for file 2 */
    char *fdb_path1;          /* pathname for file 1 */
    char *fdb_path2;          /* pathname for file 2 */
    unsigned long fdb_bn;     /* block number */
    unsigned long fdb_fixable; /* fault fixable indicator */
} FAULT_DESC_BLOCK;
```

This structure cannot be packed. The contents of the `fault_desc_block` fields are as follows:

<code>fdb_code</code>	Contains a fault code describing the type of fault.
<code>fdb_fn1</code>	Contains the file number of the file.
<code>fdb_fn2</code>	For faults involving two files, contains the file number of the second file.
<code>fdb_path1</code>	Contains a pointer to the file's complete pathname. This pathname is constructed by <code>verify_vol()</code> within the <code>verify_vol()</code> work area.
<code>fdb_path2</code>	For faults involving two files, contains a pointer to the second file's complete pathname.
<code>fdb_bn</code>	For faults involving a specific block, contains the block number of the affected block.
<code>fdb_fixable</code>	Indicates whether the fault can be corrected by <code>verify_vol()</code> , as follows: <code>fdb_fixable = 0</code> means fault is <i>not</i> fixable. <code>fdb_fixable = 1</code> means fault is fixable.

Fault Types

Table 2-2 beginning on page 2-134 summarizes, for each fault type, the contents of each field. An X indicates the field is used in describing the fault. The last column indicates whether or not the fault is fixable.

NOTE: Footnotes *a* through *f* for Table 2-2 are all listed at the end of the table.

TABLE 2-2 Fault Summary

Mnemonic	Description	Hex	FN1	FN2	PATH1	PATH2	BN	Fixable
VF_BMOFL	The bitmap and FLIST ^a , as specified in ROOTBLOCK ^b , overlap.	2101						N
VF_BMSIZ	The bitmap size and volume size, as specified in ROOTBLOCK, are inconsistent with one another.	2102						N
VF_FLSIZ	The FLIST size and number of file descriptors, as specified in ROOTBLOCK, are inconsistent with one another.	2103						N
VF_BMOVL	The bitmap, as specified in ROOTBLOCK, extends beyond the end of the volume.	2104						N
VF_FLOVL	The FLIST, as specified in ROOTBLOCK, extends beyond the end of the volume.	2105						N
VF_BMDA	The bitmap, as specified in ROOTBLOCK, overlaps the volume's data area.	2106						N
VF_FLDA	The FLIST, as specified in ROOTBLOCK, overlaps the volume's data area.	2107						N
VF_BMEXT	The extent map in the bitmap FD ^c disagrees with ROOTBLOCK.	2108						N
VF_FLEXT	The extent map in the FLIST FD disagrees with ROOTBLOCK.	2109						N
VF_NDRFD	The FD for the ROOT directory does not indicate it is a directory.	210A						Y

TABLE 2-2 Fault Summary (Continued)

Mnemonic	Description	Hex	FN1	FN2	PATH1	PATH2	BN	Fixable
VF_FDMU	A FD is used by more than one file. <code>verify_vol()</code> returns the FN ^d and pathname of both files, although the FNs are the same.	210B	X	X	X	X		N
VF_FDFRE	A FD that is in use is marked free.	210C	X		X			N
VF_FDUSE	A FD that is not in use is marked as in use.	210D	X					Y
VF_NSSFD	The FD of a non-system file indicates the file is a system file.	2110	X		X			Y
VF_SNSFD	The FD for a system file (ROOTBLOCK, BITMAP, or FLIST) does not indicate the file is a system file.	2111	X		X			Y
VF_PARFD	The parent FN within a FD does not point to the file's parent directory.	2112	X		X			Y
VF_FCFD	The file count within a FD for a directory is incorrect.	2113	X		X			Y
VF_SIZEFD	A file's FD indicates that its logical size is greater than its physical size.	2114	X		X			Y
VF_ANXFD	A file has an annex size of 0 in its FD. This fault is corrected by setting the annex size to 1.	2115	X		X			Y
VF_EXTFD	See "Extent Map Faults" on page 2-138.	2118	X		X		X	N
VF_INFD	See "Extent Map Faults" on page 2-138.	2119	X		X		X	N
VF_IXFD	See "Extent Map Faults" on page 2-138.	211A	X		X		X	N
VF_TB CFD	See "Extent Map Faults" on page 2-138.	211B	X		X			Y

TABLE 2-2 Fault Summary (Continued)

Mnemonic	Description	Hex	FN1	FN2	PATH1	PATH2	BN	Fixable
VF_LLbfd	See “Extent Map Faults” on page 2-138.	211C	X		X			Y
VF_LLbin	See “Extent Map Faults” on page 2-138.	211D	X		X			Y
VF_EXTIN	See “Extent Map Faults” on page 2-138.	211E	X		X		X	N
VF_INIX	See “Extent Map Faults” on page 2-138.	211F	X		X		X	N
VF_LLbiX	See “Extent Map Faults” on page 2-138.	2120	X		X		X	Y
VF_DBDA	See “Extent Map Faults” on page 2-138.	2121	X		X		X	N
VF_INDA	See “Extent Map Faults” on page 2-138.	2122	X		X		X	N
VF_IXDA	See “Extent Map Faults” on page 2-138.	2123	X		X		X	N
VF_DfDIR	A directory contains the same filename more than once. <code>verify_vol()</code> provides the FN and pathname of both files, although in this case the pathnames are identical.	2124	X	X	X	X		N
VF_IFDIR	A directory entry contains an illegal filename. <code>verify_vol()</code> provides the FN and pathname of the file. Note that since the filename is illegal, the last filename in the pathname may not be ASCII. ^e	2125	X	X	X	X		Y
VF_FNDIR	A directory entry contains an illegal FN: one that exceeds the allowed maximum. In this case, <code>fdb_path1</code> contains the file's pathname while <code>fdb_fn1</code> contains the illegal FN. <code>fdb_fn2</code> and <code>fdb_path2</code> describe the directory containing the illegal entry. ^f	2126	X	X	X	X		Y
VF_BKMU	A single block is used by more than one file.	2128	X	X	X	X	X	N

TABLE 2-2 Fault Summary (Continued)

Mnemonic	Description	Hex	FN1	FN2	PATH1	PATH2	BN	Fixable
VF_BBUSE	A bad block is in use.	2129	X		X		X	N
VF_BKFRE	A block that is in use is also marked as free in the volume bitmap.	212A	X		X		X	Y
VF_BBFRE	A bad block is marked as free in the volume bitmap.	212B					X	Y
VF_BKUSE	An unused block is marked as in use in the volume bitmap.	212C					X	Y
VF_INSUFF	Work area too small.	2200						Y
VF_MAXDEPTH	Directory depth exceeds maximum.	2201						Y
VF_ABORT	Verify routine aborted by user.	2202						Y

- a. The file descriptor list, one of the management blocks described in the pHILE+ chapter of *System Concepts*.
- b. The root block, one of the management blocks described in the pHILE+ chapter of *System Concepts*.
- c. File descriptor.
- d. File number.
- e. If `faultp()` so requests, `verify_vol()` corrects this fault by changing the filename to `VFN_00000000`, where `00000000` is the hexadecimal representation of the file's FN. For example, if the file has an FN of 29 (decimal), the filename is set to `VFN_0000001D`.
- f. If `faultp()` so requests, `verify_vol()` corrects this fault by setting the FN to zero (that frees the entry for reuse).

Extent Map Faults

The faults listed in Table 2-3 involve errors in the extent map of a particular file. Recall that the extent map consists of:

- Up to 10 extent descriptors within the file's FD.
- Within the file's FD, an indirect block descriptor that describes an indirect block containing additional extent descriptors.
- Within the file's FD, an index block descriptor that describes an index block containing additional indirect block descriptors.

`verify_vol()` checks for illegal blocks both within an extent and within an indirect or index block descriptor. A block is illegal if its block number is equal to or greater than the number of blocks on the volume. For example, on a volume containing 1000 blocks, any block number greater than 999 is illegal.

A file can be viewed as a sequence of logical blocks numbered from 0. For example, on a volume with 1K blocks, a 4.3 Kbyte file would consist of logical blocks 0 through 4 (logical block 4 being only partly filled.)

Every FD, every indirect block descriptor, and every index block descriptor contains a *last logical block* (LLB) field that indicates the largest logical block number addressed by the associated structure. For example, an indirect block descriptor may have LLB = 200, meaning that the last block in the last extent in the indirect block is the 200th block in the file. `verify_vol()` checks the LLB of every FD, indirect block descriptor, and index block descriptor and reports any inconsistencies.

TABLE 2-3 Extent Map Faults

VF_EXTFD	An FD contains an extent containing an illegal block.
VF_INFDD	An FD contains an illegal indirect block number.
VF_IXFD	An FD contains an illegal index block number.
VF_TBDFD	The block count within the FD conflicts with the actual number of blocks in the file.
VF_LLDFD	The LLB in the FD (for the first 10 extents) is incorrect.
VF_LLBDIN	The LLB within an indirect block descriptor within an FD is incorrect.

TABLE 2-3 Extent Map Faults

VF_EXTIN	An indirect block contains an extent containing an illegal block.
VF_INIX	An index block contains an illegal indirect block number.
VF_LLBIIX	Within an index block, the LLB associated with an indirect block is incorrect.
VF_DBDA	A directory block resides in the data area of the volume.
VF_INDA	An indirect block resides in the data area of the volume.
VF_IXDA	An index block resides in the data area of the volume.

Bad Blocks

A bad block is a block that cannot be read and/or written and therefore cannot be used by pHILE+ file system manager. There are a number of possible strategies for handling bad blocks. One strategy is to mask or redirect them at the driver level so that pHILE+ file system manager never sees them. Another method, which is described here in detail, involves “mapping out” those blocks in the volume’s bitmap, so that they are never allocated by pHILE+ file system manager. `verify_vol()` facilitates such modifications to the bitmap.

Recall that each volume contains a bitmap describing which blocks on the volume are in use, and which are free. If the corresponding bit in the map is set to 1, the block is considered to be in use; otherwise, it is considered to be available for allocation by the pHILE+ file system manager when needed. If the bit corresponding to a bad block can be set to 1 before the block is allocated, then the pHILE+ file system manager will never allocate the block, and hence will never read or write it.

To facilitate bad block handling, `verify_vol()` accepts as an input parameter a list of bad blocks. When examining the volume’s bitmap, `verify_vol()` expects bits corresponding to these bad blocks to be set to 1, while at the same time expecting the block to be unused. If a bad block is in use, or its corresponding bit is not set, a fault is generated.

The remainder of this section gives a brief outline of a recommended method for handling bad blocks.

There are two types of bad blocks:

Dead Blocks — These blocks are known to be bad prior to volume initialization. They are normally the result of manufacturing defects. Typically, the device

manufacturer provides a list of such blocks with each device. They can also be detected by testing the device prior to its initialization.

Failed Blocks — These are blocks that fail some time after the volume has been initialized. They are normally detected in the course of reading or writing the affected block.

Dead blocks are much simpler to handle than failed blocks, because they are detected *before* the pHILE+ file system manager has allocated them. To handle these blocks, perform the following steps:

1. Initialize the volume, taking care not to place the bitmap or FLIST onto any dead blocks (Note: blocks 2 and 3 must not be dead.)
2. Mount the volume and call `verify_vol()`, providing a bad block list containing all dead blocks.
3. Have `faultp()` always return `status = 1` (correct fault and continue) for `fdb_code == VF_BBFRE` so that `verify_vol()` will mark the blocks as in use (note that the "bad block is marked free" error is correctable).

Failed blocks are harder to handle because the block was already allocated by pHILE+ file system manager before it failed. The block may or may not contain valid data, depending on exactly when the failure occurred. However, since the block is allocated, its corresponding bit is already set. To eliminate such bad blocks, perform the following steps:

1. Add the failed block to the existing list of bad blocks.
2. Invoke `verify_vol()`, which will report `VF_BBUSE`, bad block is in use by a particular file.
3. By whatever means, salvage as much of the file as possible, and then delete the file. This returns the bad block to the free block pool and clears its corresponding bit.
4. Invoke `verify_vol()` again. `verify_vol()` now reports `VF_BBFRE`, "bad block is marked free". Now have `faultp()` use `return status = 1`, to mark the bad block as unavailable for allocation.

Step 3 may be complicated. For example, if a bad block occurs within a directory page, then the entire directory must be deleted after saving as much of its contents as possible.

Note that if `verify_vol()` is to be used to maintain the bitmap in this manner, then an updated list of all bad blocks on the volume must be kept. Integrated Systems suggests that you store the bad block list itself as a file on the volume.

See Also

`mount_vol`

write_f Writes to an open file.

```
#include <phile.h>
unsigned long write_f(
    unsigned long fid,    /* file identifier */
    void *buffer,         /* output buffer */
    unsigned long bcount /* output byte count */
)
```

Volume Types

All volume types except CD-ROM.

Description

`write_f()` writes data into a file. It begins at the current position of the connection's `L_ptr`.

After `write_f()`, the file's `L_ptr` is updated to point to the byte after the last byte written.

This call overwrites the original content of the file. If necessary, `write_f()` expands the file by allocating space to hold the written data.

Arguments

<code>fid</code>	Specifies the file identifier associated with the file.
<code>buffer</code>	Points to the data to write.
<code>bcount</code>	Specifies the number of bytes to write.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x2003	E_BADVOL	Inconsistent data on volume; volume corrupted.
0x2015	E_RO	Requested operation not allowed on this file.
0x201A	E_FIDBIG	Invalid FID; exceeds maximum.
0x201B	E_FIDOFF	Invalid FID; file closed.
0x201C	E_ININFULL	Index block is full.
0x201D	E_VFULL	Volume is full.
0x2022	E_LOCKED	Data is locked.
0x2052	E_EREMOTE	Too many levels of remote in path.
0x2054	E_EIO	A hard error happened at remote site.
0x2055	E_EACCES	Task does not have access permissions.
0x2056	E_EISDIR	Illegal operation on a directory.
0x2057	E_EQUOT	Quota exceeded.
0x2058	E_ESTALE	Stale NFS file handle.
0x205B	E_ENXIO	No such device or address.
0x205C	E_ENODEV	No such device.
0x2060	E_BADCD	CD-ROM volume; illegal operation.
0x2070	E_EAUTH	RPC authorization is not available.
0x2071	E_ENFS	Portmap failure on the host.
0x2072	E_ETIMEOUT	NFS call timed out.
0x2074	E_ENOAUTHBLK	No RPC authorization blocks are available.
0x2075	E_ECANTSEND	Failure in sending call.
0x2076	E_ECANTRECV	Failure in receiving result.
0x2077	E_PROBUNAVAIL	Program not available.
0x2078	E_EPROGVERSMISMATCH	Program version mismatched.

Hex	Mnemonic	Description
0x2079	E_ECANTDECODEARGS	Decode arguments error.
0x207A	E_EUNKNOWNHOST	Unknown host name.
0x207B	E_EPROGNOTREGISTERED	Remote program is not registered.
0x207C	E_EUNKNOWNPROTO	Unknown protocol.
0x207D	E_EINTR	Call interrupted.
0x207E	ERPC	All other RPC errors.

Notes

1. On pHILE+ and MS-DOS volumes, `write_f()` operations are more efficient if `bcount` is an integral multiple of the block size and the `L_ptr` is positioned at a block boundary.
2. On pHILE+ and MS-DOS volumes, if the requested data includes either entire blocks or a contiguous sequence of blocks and if such blocks are not already in the buffer cache, the pHILE+ file system manager writes these blocks directly from the user's buffer (without going through the buffer cache).
3. `write_f()` automatically positions the `L_ptr` for sequential write operations. If random writes are needed, the `lseek_f()` call should be used to reposition the `L_ptr`.
4. Writing to system or directory files is not allowed.
5. `write_f()` expands a file if space is needed to accommodate the new data.
6. CD-ROM volumes are read-only.

See Also

`lseek_f`, `sync_vol`, `write_vol`

write_vol	Writes directly to a pHILE+ formatted volume.
------------------	---

```
#include <phile.h>
unsigned long write_vol(
    char *device,          /* volume name */
    unsigned long block,   /* base block */
    unsigned long index,   /* byte offset */
    unsigned long bcount,  /* number of bytes to write */
    void *buffer           /* output buffer */
)
```

Volume Types

pHILE+ and MS-DOS formatted volumes.

Description

`write_vol()` writes data directly to a pHILE+ formatted volume (bypassing the file system organization imposed by the pHILE+ file system manager).

Arguments

<code>device</code>	Points to the null-terminated name of the volume to read.
<code>block</code>	Specifies the logical block number where writing begins.
<code>index</code>	Specifies where to begin writing within the specified block.
<code>bcount</code>	Specifies the number of bytes to write.
<code>buffer</code>	Points to the memory area containing the data to write.

Return Value

This system call returns 0 on success or an error code on failure.

Error Codes

Hex	Mnemonic	Description
0x200A	E_DMOUNT	Volume not mounted.
0x2015	E_RO	Operation not allowed on read-only system files, directories, or mounted volumes.
0x2025	E_IDN	Illegal device name.
0x2050	E_BADNFS	NFS volume; illegal operation.
0x2060	E_BADCD	CD-ROM volume; illegal operation.

Notes

1. If `index` is larger than the volume's block size, the write begins in a subsequent block. For example, on a volume with a 1024-byte block size, writing block 5, index 1224, is the same as writing block 6, index 200.
2. `write_vol()` does not check for the end of the volume; blocks beyond the specified volume size can be written if they physically exist.
3. If the requested data includes either entire blocks or a contiguous sequence of blocks and if such blocks are not already in the buffer cache, the pHILE+ file system manager writes the blocks directly to the volume (without going through the buffer cache.) Therefore, `write_vol()` operations are more efficient when `bcount` and `index` equal integral multiples of blocks.
4. `write_vol()` execution on any block is allowed, including blocks in system files. Therefore, use this call cautiously.
5. CD-ROM volumes are read-only.

See Also

`read_vol`

3

pREPC+ System Calls

3

This chapter provides detailed information on each system call in the pREPC+ component of pSOSystem. The calls are listed alphabetically, with a multipage section of information for each call. Each call's section includes its syntax, a detailed description, its arguments, and its return value. Where applicable, the section also includes the headings "Notes" and "See Also." "Notes" provides important information not specifically related to the call's description, and "See Also" indicates other calls that have related information.

If you need to look up a system call by its functionality, refer to Appendix A, "Tables of System Calls," which lists the calls alphabetically by component and provides a brief description of each call.

pREPC+ error codes are listed in Appendix B, "Error Codes." For practical reasons, they are not listed here, because every pREPC+ system call can return most or all of the pREPC+ error codes. In addition, errors in other pSOSystem components or device drivers can be reported by pREPC+ system calls.

abort Aborts a task.

```
#include <stdlib.h>
void abort (void);
```

Description

The `abort()` macro is used to terminate a task. `abort()` simply invokes the `exit()` macro with an argument of zero. For further details, refer to the `exit()` macro on page 3-28.

Return Value

If the task is successfully deleted, the `abort()` macro does not return to its caller. If the task cannot be deleted successfully, `abort()` suspends the task indefinitely and does not return to its caller unless the task is explicitly resumed by another task in the system.

Error Codes

None.

Notes

Callable From

- Task

See Also

`exit()`

abs Computes the absolute value of an integer.

```
#include <stdlib.h>
int abs (
    int j    /* long integer */
)
```

Description

The `abs()` function converts the integer `j` into its absolute value. If the result cannot be represented, the behavior is undefined.

Arguments

`j` Specifies the integer to be converted.

Return Value

`abs()` returns the absolute value.

Error Codes

None.

Notes

Callable From

- Task
- ISR

See Also

`labs`

asctime Converts the broken-down time to a string.

```
#include <time.h>
char *asctime (
    const struct tm *timeptr    /* broken-down time */
)
```

Description

The function `asctime()` converts the broken-down time pointed to by `timeptr` to an equivalent string representation of the form:

```
Sun Jan 1 12:30:13 1995\n\0
```

Arguments

`timeptr` Points to a `tm` structure that stores the broken-down time. The `tm` structure is defined in the `mktime()` description on page 3-111.

Return Value

The `asctime()` function returns a pointer to the calendar time string.

Error Codes

Refer to Appendix B.

Notes

This function is non-reentrant as it returns a pointer to a statically allocated data area. The reentrant version of this function is `asctime_r()`.

Callable From

- Task

See Also

`asctime_r`, `ctime`, `mktime`, `time`

asctime_r (Reentrant) Converts the broken-down time to a string.

```
#include <time.h>
char *asctime_r (
    const struct tm *timeptr, /* pointer to broken-down time */
    char *buf,                /* result buffer */
    int buflen                /* result buffer length */
)
```

Description

asctime_r() is the reentrant version of the ANSI function asctime(), as defined by POSIX 1003.1c. It converts the broken-down time pointed to by timeptr to an equivalent string representation of the form:

```
Sun Jan 1 12:30:13 1995\n\n0
```

and stores the string in the buffer pointed to by buf, which is assumed to have space for at most buflen characters. An error may be returned if the converted string contains more than buflen characters.

Arguments

timeptr	Points to a structure of type tm that stores the broken-down time. The tm structure is defined in the mktime() description on page 3-111.
buf	Points to the buffer where asctime_r() stores the result.
buflen	Specifies the size of buf.

Return Value

Upon success, asctime_r() returns the value of buf. On failure, it returns NULL and sets errno.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

`asctime`, `ctime`, `ctime_r`, `mktime`, `time`

assert Verifies that a program is operating correctly.

```
#include <assert.h>
void assert (
    int expression      /* test expression */
)
```

Description

The `assert()` macro, defined in the header file `assert.h`, writes error information to `stderr` if the expression `expression` evaluates to zero. The error information includes the text of the argument, the name of the source file, and the source line number. The last two of these are respectively the values of the preprocessing macros `__FILE__` and `__LINE__`.

If `expression` does not evaluate to zero, `assert()` does nothing.

Arguments

`expression` Specifies the expression to be evaluated.

Return Value

The `assert()` macro returns no value.

Error Codes

None.

Notes

Callable From

- Task

atof Converts a string to a double.

```
#include <stdlib.h>
double atof(
    const char *nptr    /* string */
)
```

Description

This function converts the initial part of the string pointed to by `nptr` to a double representation. Leading white spaces are ignored. The argument `nptr` can be in scientific exponential form (for example, +123.45e+67, -123.45E+67). This function stops parsing `nptr` when it detects a character inconsistent with a double data type. If the first nonwhite space character is other than a sign, a digit or a decimal point, a value of 0 is returned.

Except for the behavior on error, this call is equivalent to:

```
strtod(str, (char **)NULL);
```

Arguments

`nptr` Points to the string to be converted.

Return Value

This function returns the converted value. In the event of an error, `errno` is set to indicate the condition.

Error Codes

Refer to Appendix B.

Notes

The pREPC+ library returns double values (including floating point) in the CPU register pair designated by the compiler to receive a return value of type double from a function call when a hardware floating point is *not* selected. Please refer to your compiler manual for the register pair. Additionally, if the FPU bit is set in the processor type entry of the Node Configuration Table, the pREPC+ library also

places the floating point value in the floating point register designated by the compiler to receive a return value of type `double` when a hardware floating point *is* selected.

Callable From

- Task

See Also

`strtod`

atoi

Converts a string to an integer.

```
#include <stdlib.h>
int atoi(
    const char *nptr    /* string */
)
```

Description

The `atoi()` function converts the initial part of the string pointed to by `nptr` to an `int` representation. Leading white spaces are ignored. The conversion terminates when a nondigit character is detected. If the first nonwhite space character is not a digit, a value of 0 is returned.

Except for the behavior on error, this call is equivalent to:

```
(int) strtol(str, (char **)NULL, 10);
```

Arguments

`nptr` Points to the string to be converted.

Return Value

This function returns the converted value. If an error occurs, `errno` is set to indicate the condition.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

`strtol`

atol Converts a string to a long integer.

```
#include <stdlib.h>
long atol(
    const char *nptr    /* string */
)
```

Description

The `atol()` function converts the initial part of the string pointed to by `nptr` to a long int representation. Leading white spaces are ignored. The conversion terminates when a nondigit character is detected. If the first non-whitespace character is not a digit, a value of 0 is returned.

Except for the behavior on error, this call is equivalent to:

```
strtol(str, (char **)NULL, 10);
```

Arguments

`nptr` Points to the string to be converted.

Return Value

This function returns the converted value. If an error occurs, `errno` is set to indicate the condition.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

`strtol`

bsearch

Searches an array.

```
#include <stdlib.h>
void *bsearch(
    const void *key,      /* search key */
    const void *base,     /* start point */
    size_t nmemb,         /* number of members */
    size_t size,          /* member size */
    int (*compar)(const void *, const void *)
    /* comparison operator */
)
```

3

Description

The `bsearch()` function searches an array of `nmemb` objects, the initial element of which is pointed to by `base`, for an element that matches the object pointed to by `key`. The size of each element of the array is specified by `size`.

The array must be sorted in ascending order. A user supplied comparison function, `compar`, is called by `bsearch` with two arguments. The first argument to `compar` is a pointer to the `key` object, and the second is a pointer to an array member. The `compar` function must return an integer that is either less than, equal to, or greater than zero if `key` object is considered, respectively, to be less than, equal to, or greater than the array member.

Arguments

<code>key</code>	Points to the object to be matched in the search.
<code>base</code>	Points to the beginning of the array to be searched.
<code>nmemb</code>	Specifies the number of members in the array.
<code>size</code>	Specifies the size of each member in the array.

Return Value

This function returns a pointer to the first matching member detected. If no match is found, it returns a null pointer. In the event of an error, `errno` is set to indicate the condition.

Error Codes

Refer to Appendix B.

Notes

The `compar` function can call a limited set of pREPC+ functions. These functions consist of all character handling functions and all string handling functions except `strtok()`. Other pREPC+ functions cannot be called from `compar`.

Callable From

- Task
- ISR

See Also

`qsort`

calloc Allocates memory.

```
#include <stdlib.h>
void *calloc(
    size_t nmemb,    /* number of allocation units */
    size_t size      /* size of allocation unit */
)
```

3

Description

The `calloc()` function allocates memory for `nmemb` data objects, each of whose size is specified by `size`. The allocated memory is initialized to 0.

Arguments

<code>nmemb</code>	Specifies the number of data objects for which <code>calloc()</code> allocates memory.
<code>size</code>	Specifies the size of each data object.

Return Value

This function returns a pointer to the memory allocated, or a null pointer if no memory is allocated. If an error occurs, `errno` is set.

Error Codes

Refer to Appendix B.

Notes

1. `calloc()` calls the pSOS+ region manager to allocate the memory.
2. Memory is always allocated from Region 0.
3. The caller can be blocked if memory is not available and the wait option is selected in the pREPC+ Configuration Table.

Callable From

- Task

See Also

`free`, `malloc`, `realloc`

clearerr

Clear's a stream's error indicators.

```
#include <stdarg.h>
#include <stdio.h>
void clearerr(
    FILE *stream    /* stream pointer */
)
```

3

Description

The `clearerr()` function clears the end-of-file and error indicators for the stream pointed to by `stream`.

Arguments

`stream` Points to an open pREPC+ stream.

Return Value

This function does not return a value. If an error occurs, `errno` is set.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

ctime

Converts the calendar time to a string.

```
#include <time.h>
char *ctime (
    const time_t *timer    /* calendar time */
)
```

Description

The `ctime()` function converts the calendar time pointed to by `timer` to a string representation of the form:

Sun Jan 1 12:30:13 1995\n\n0

The time is represented in local time. This call is equivalent to:

`asctime(localtime(timer))`

The calendar time is generally obtained through a call to `time()`.

The buffer used by `ctime()` to hold the formatted output string is a statically allocated character array and is overwritten each time the function is called. To save the contents of the string, you need to copy it elsewhere.

Arguments

`timer` Points to the calendar time.

Return Value

The `ctime()` function returns the pointer to the converted calendar time string.

Error Codes

Refer to Appendix B.

Notes

This function is non-reentrant as it returns a pointer to a statically allocated data area. The reentrant version of this function is `ctime_r()`.

Callable From

- Task

See Also

`asctime`, `asctime_r`, `ctime_r`, `mktime`, `time`

ctime_r (Reentrant) Converts the calendar time to a string.

```
#include <time.h>
char *ctime_r (
    const time_t *timer,      /* calendar time */
    char *buf,                /* result buffer */
    int buflen                /* result buffer length */
)
```

Description

ctime_r() is the reentrant version of the ANSI function ctime(), as defined by POSIX 1003.1c. It converts the calendar time pointed to by timer to a string representation of the form:

Sun Jan 1 12:30:13 1995\n\n0

The time is represented in local time. ctime_r() stores the string in the buffer pointed to by buf, which is assumed to have space for at most buflen characters. An error may be returned if the converted string contains more than buflen characters.

The calendar time is generally obtained through a call to time().

Arguments

timer	Points to the calendar time.
buf	Points to the buffer where ctime_r() stores the result.
buflen	Specifies the size of buf.

Return Value

Upon success, asctime_r() returns the value of buf. On failure, it returns NULL and sets errno.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

`asctime`, `asctime_r`, `ctime`, `mktime`, `time`

difftime Computes the difference between two calendar times.

```
#include <time.h>
double difftime (
    time_t time1,    /* finish time */
    time_t time0     /* start time */
)
```

Description

The `difftime()` function computes the difference, in seconds, between two calendar times: `time1 - time0`.

Arguments

<code>time1</code>	Specifies the finish time.
<code>time0</code>	Specifies the start time.

Return Value

The `difftime()` function returns the difference expressed in seconds as a double.

Error Codes

None.

Notes

Callable From

- Task

See Also

`time`

div Performs a division operation on two specified integers.

```
#include <stdlib.h>
div_t div (
    int number,      /* numerator */
    int denom        /* denominator */
)
```

3

Description

The `div()` function computes the quotient and remainder of the division of the numerator `number` by the denominator `denom`. If the division is inexact, the resulting quotient is the integer of lesser magnitude that is the nearest to the algebraic quotient. If the result cannot be represented, the behavior is undefined; otherwise, `quot * denom + rem` is equal to `number`.

Arguments

<code>number</code>	Specifies the numerator.
<code>denom</code>	Specifies the denominator.

Return Value

The `div()` function returns a structure of type `div_t`. This structure is defined in `stdlib.h` as follows:

```
typedef struct {
    int quot; /* the quotient */
    int rem;  /* the remainder */
} div_t;
```

Error Codes

None.

Notes

Callable From

- Task
- ISR

See Also

ldiv

errno The error number returned by the last failing system call.

```
#include <errno.h>
int errno;
```

Description

The `errno` is a macro which expands to a modifiable lvalue that has type `int`. Its value can be set to a positive number by several library or system calls. The macro is defined in the header file `errno.h`. It returns the error number from the last failing system call or library function.

If the macro definition is suppressed (by using `#undef` pre-processor directive), or an application defines an identifier with the name `errno`, the behavior is undefined.

The value of `errno` is zero at task startup, but is never set to zero by any library function or system service.

Return Value

This macro returns the current value of `errno` for the calling task.

Error Codes

Refer to Appendix B.

Notes

`errno` generates a call to the pSOS+ `errno_addr()` system service.

Callable From

- Task

exit Terminates a task.

```
#include <stdlib.h>
void exit(
    int status    /* termination status */
);
```

Description

The `exit()` macro terminates the task that invokes it. `exit()` prints an error message on the task's standard error stream if a non-zero `status` is passed to it. It then executes the task exit sequence described in `t_delete()` on page 1-139, with the exception that it does not automatically invoke “close” functions for pSOSystem components other than pREPC+ (see Note). If the task cannot be deleted successfully, `exit()` suspends the task indefinitely.

Arguments

`status` Contains the termination status printed by the error message.

Return Value

If the task is successfully deleted, the `exit()` macro does not return to its caller. If the task cannot be deleted successfully, `exit()` suspends the task indefinitely and does not return to its caller unless the task is explicitly resumed by another task in the system.

Error Codes

None.

Notes

If you have pSOSystem components other than pREPC+ configured in your system, you need to edit the definition of `exit()` in `stdlib.h` to uncomment the necessary “close” functions in the task exit sequence. The “close” functions release any task-specific resources held by pSOSystem components.

Callable From

- Task

See Also

`abort`

fclose Closes a stream.

```
#include <stdarg.h>
#include <stdio.h>
int fclose(
    FILE *stream    /* stream pointer */
)
```

Description

The `fclose()` function first flushes the buffer associated with the stream pointed to by `stream` and closes the associated file or I/O device. Any unwritten buffered data is written to the associated file or I/O device, and any unread buffered data is discarded. The stream is disassociated from the file or I/O device.

If the buffer was automatically allocated when the stream was opened, it is reclaimed by the system. The user is responsible for returning user-supplied buffers.

When invoked with a null stream pointer, `fclose()` has a special significance under pREPC+. It causes pREPC+ to:

- Close all streams opened by the calling task.
- Reclaim all memory allocated by pREPC+ on behalf of the calling task, either implicitly or explicitly by calls to `malloc()` or `calloc()` functions.

Arguments

`stream` Points to an open pREPC+ stream.

Return Value

This function returns 0 if successful or end-of-file (EOF) if an error occurs. If an error occurs, `errno` is set.

Error Codes

Refer to Appendix B.

Notes

1. pREPC+ calls the pSOS+ function `de_close()` if the stream was associated with an I/O device.
2. pREPC+ calls the pHILE+ function `close_f()` if the stream was associated with a disk file.
3. If a task uses any of the pREPC+ functions, it must call `fclose(0)` to release all pREPC+ resources prior to deleting itself through a `t_delete()` system call. Refer to the `t_delete()` call description on page 1-138 for the complete exit sequence.

Callable From

- Task

See Also

`fopen`

feof Tests a stream's end-of-file indicator.

```
#include <stdarg.h>
#include <stdio.h>
int feof(
    FILE *stream    /* stream pointer */
)
```

Description

The `feof()` function tests the end-of-file indicator for the stream pointed to by `stream`.

Arguments

`stream` Points to an open pREPC+ stream.

Return Value

This function returns a nonzero number if the end-of-file indicator is set for `stream` and zero if it is not set.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

ferror	Tests a stream's error indicator.
---------------	-----------------------------------

```
#include <stdarg.h>
#include <stdio.h>
int ferror(
    FILE *stream    /* stream pointer */
)
```

Description

This function tests the error indicator for the stream pointed to by `stream`.

Arguments

stream	Points to an open pREPC+ stream.
--------	----------------------------------

Return Value

This function returns a nonzero number if the error flag is set and zero if it is not set.

Error Codes

Refer to Appendix B.

Notes

Callable From

- ## ■ Task

fflush

Flushes the buffer associated with an open stream.

```
#include <stdarg.h>
#include <stdio.h>
int fflush(
    FILE *stream    /* stream pointer */
)
```

Description

The `fflush()` function writes any unwritten, buffered data associated with the stream pointed to by `stream` to the file or I/O device. An error is returned if the stream has not been opened for write or update. If `stream` is a null pointer, the `fflush()` function performs the flushing action on all the streams open for write or update.

Arguments

`stream` Points to an open pREPC+ stream.

Return Value

This function returns 0 if successful or EOF on error. If an error occurs, `errno` is set.

Error Codes

Refer to Appendix B.

Notes

1. If the write is to an I/O device, `fflush()` calls the pSOS+ I/O call `de_write()`.
2. If the write is to a disk file, `fflush()` calls the pHILE+ call `write_f()`.

Callable From

- Task

fgetc

Gets a character from a stream.

```
#include <stdarg.h>
#include <stdio.h>
int fgetc(
    FILE *stream    /* stream pointer */
)
```

3

Description

The `fgetc()` function reads the next character, as an unsigned char converted to an int, from the input stream pointed to by `stream` and advances the associated file position indicator for the stream, if defined. It is operationally equivalent to the `getc` function.

Arguments

`stream` Points to an open pREPC+ stream.

Return Value

This function returns the character that is read from the stream. If the end-of-file condition is detected, the stream's end-of-file indicator is set and `EOF` is returned. If a read error occurs, the stream's error indicator is set, `EOF` is returned, and `errno` is set.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

fgetpos

Gets the current file position indicator for `fsetpos`.

```
#include <stdarg.h>
#include <stdio.h>
int fgetpos(
    FILE *stream,    /* stream pointer */
    fpos_t *pos      /* stream position */
)
```

Description

The `fgetpos()` function stores the current value of the file position indicator for the stream pointed to by `stream` in the object pointed to by `pos`. This value can be used by the `fsetpos()` function to reposition the file position indicator of the stream to its position at the time of the call to the `fgetpos()` function.

Arguments

<code>stream</code>	Points to an open pREPC+ stream.
<code>pos</code>	Points to the object where <code>fgetpos()</code> stores the current file position indicator.

Return Value

If successful, this function returns a zero. If not successful or if `stream` references an I/O device, this function returns an EOF and sets `errno`.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

fgets

Gets a string from a stream.

```
#include <stdarg.h>
#include <stdio.h>
char *fgets(
    char *s,          /* buffer */
    int n,            /* length */
    FILE *stream      /* stream pointer */
)
```

3

Description

The `fgets()` function reads at most one less than the number of characters specified by `n` from the stream pointed to by `stream`. The characters go into the user buffer pointed to by `s`. The function stops reading characters when a new-line character (which is retained) or an end-of-file condition is detected. A null character is written immediately after the last character is read into the user buffer.

If `stream` references a disk file, its position indicator is advanced.

Arguments

<code>s</code>	Points to the user buffer where <code>fgets()</code> stores characters.
<code>n</code>	Specifies the number of characters to read, plus one for the null terminator.
<code>stream</code>	Points to an open pREPC+ stream.

Return Value

This function returns `s` if successful. If a read error occurs or an end-of-file condition is detected before any characters are read, a null pointer is returned and `errno` is set.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

fputs

fopen Opens a file.

```
#include <stdarg.h>
#include <stdio.h>
FILE *fopen(
    const char *filename,    /* file name */
    const char *mode         /* access mode */
)
```

3

Description

The `fopen()` function opens a disk file or I/O device whose name is specified by the string pointed to by `filename` and associates a stream with it.

`fopen()` allocates a `FILE` structure for the opened stream. It contains control information for the opened I/O device or disk file. `fopen()` returns a pointer to the allocated `FILE` structure that subsequent calls require to perform various I/O operations on the I/O device or disk file (for example, for an `fread` or `fwrite` call).

Disk files have position indicators that determine where the next byte is read from or written to in the file. Position indicators have no meaning for I/O devices.

Arguments

<code>filename</code>	Points to the name of the disk file or I/O device to be opened.
<code>mode</code>	Points to a string that specifies the mode in which the file is to be opened. The <code>mode</code> string must begin with one of the following sequences:
<code>r</code>	Open text file for reading.
<code>w</code>	Truncate to zero length or create text file for writing.
<code>a</code>	Append: open or create text file for writing at the end-of-file.
<code>rb</code>	Open binary file for reading.
<code>wb</code>	Truncate to zero length or create binary file for writing.

<code>ab</code>	Append: open or create a binary file for writing at the end-of-file.
<code>r+</code>	Open text file for updating, reading and writing at the current file position.
<code>w+</code>	Truncate to zero length or create text file for updating, reading and writing at the current file position.
<code>a+</code>	Append: open or create text file for updating, reading at the current file position and writing at the end-of-file.
<code>r+b</code> or <code>rb+</code>	Open binary file for updating, reading and writing at the current file position.
<code>w+b</code> or <code>wb+</code>	Truncate to zero length or create binary file for updating, reading and writing at the current file position.
<code>a+b</code> or <code>ab+</code>	Append: open or create a binary file for updating, reading at current file position and writing at the end-of-file.

Opening a disk file with read mode (`r` as the first character in `mode` argument) fails if the file does not exist or cannot be read.

Opening a disk file with append mode (`a` as the first character in `mode` argument) causes all subsequent writes to the then current EOF, regardless of intervening calls to the `fseek` function.

When a disk file is opened with update mode (`+` as the second or third character in `mode` argument) both read and write operations may be performed on the associated stream. However, output may not be directly followed by input, or vice-versa, without an intervening call to the `fflush` function or to a file positioning function, via `fseek`, `fsetpos` and `rewind`. The only exception to the above rule is a read operation that encounters end-of-file.

Return Value

If the open operation is successful, this function returns a pointer to the `FILE` object that must be used in subsequent calls to specify this opened stream. If the operation fails, a null pointer is returned and `errno` is set.

Error Codes

Refer to Appendix B.

Notes

1. The disk files are managed by the pHILE+ file system manager and are designated by pHILE+ pathnames (for example, 0.1/abc). The I/O devices are identified by pSOS+ logical device numbers represented as a string of the form M.N, where M is the major number and N is the minor number of the I/O device (for example, 0.1). When reading or writing disk files, the pREPC+ library calls the pHILE+ file system manager. When reading or writing I/O devices, the pREPC+ library calls the pSOS+ I/O Supervisor directly.
2. `fopen()` internally makes a call to the pHILE+ `open_f` function when it opens a disk file and the pSOS+ `de_open` function when it opens an I/O device.
3. If the volume is an NFS volume, two conditions can cause problems related to the file mode. The conditions are as follows:

- a. A file that did not previously exist is created in an NFS volume by the `fopen()` call with a UNIX-like privilege mode automatically set to 0x180 (octal 600/rw for the user.) If, for example, the mode in the `fopen()` call is manually specified as `w`, a conflict could result. The pREPC+ library allows only file operations that are valid for the specified mode. Therefore (using the preceding example), the pREPC+ library does not allow a read operation on the following file:

```
fopen("0.0/file1.dat", "w");
```

- b. The restrictions placed on file operations depend on the mode extended to files that exist prior to the `fopen()` call. For example:

```
fopen("0.0/file2.dat", "r");
```

succeeds if `file2.dat` exists prior to the `fopen()` call. A subsequent read operation would fail if that file had an access mode (under NFS) that did not allow a user-read. Currently, no method exists under the pREPC+ library to change the access mode of an NFS file.

4. Since the underlying mechanism (pHILE+ and pSOS+ I/O device manager) do not, as of yet, support the concept of text files, pREPC+ treats text files as binary files. However, for forward compatibility, you must specify the "b" character in the mode string if the file being opened through `fopen` contains

binary data that should be read or written without performing any translations on it.

5. Though native MS-DOS systems differentiate between text and binary files, the pHILE+ implementation of MS-DOS file system does not.
6. For I/O devices, the text streams are treated as binary streams.
7. For I/O devices, the operations of truncating, creating and appending are meaningless. In modes starting with `w` or `a`, an error is returned if the device being opened is not configured into the system. Also, the append mode is treated the same as the write mode for I/O devices.

Callable From

- Task

See Also

`fclose`, `fseek`

fprintf

Prints formatted output to a stream.

```

#include <stdarg.h>
#include <stdio.h>
int fprintf(
    FILE *stream,          /* stream pointer */
    const char *format,    /* format control */
    ...                    /* arguments 1 through n */
)

```

3

Description

The `fprintf()` function writes output to the stream pointed to by `stream`. A format control string, pointed to by `format`, specifies how the subsequent arguments are converted for output.

Arguments

- | | |
|---------------------|--|
| <code>stream</code> | Points to an open pREPC+ stream. |
| <code>format</code> | <p>Points to the format control string. The <code>format</code> string consists of ordinary characters (except the <code>%</code> character) and conversion specifications. The ordinary characters are simply copied to the output stream. The conversion specifications determine the form of the arguments' output. Each argument should have one conversion specification.</p> <p>Each conversion specification begins with a <code>%</code> character and ends with a conversion specification character. One or more of the following can be positioned between the <code>%</code> and ending specification character, in the order specified below:</p> <ul style="list-style-type: none"> ■ Zero or more <i>flags</i> (in any order) that modify the meaning of the conversion specification. ■ An optional minimum <i>field width</i>. If the converted value has fewer characters than the field width, it will be padded with spaces (by default) on the left (or right, if the left adjustment flag, described below, has been given) to the field width. The field width takes the form of an asterisk <code>*</code> (described below) or a decimal integer. |

- An optional *precision* that gives the minimum number of digits to appear for the d, i, o, u, x, and X conversions, the number of digits to appear after the decimal-point character for e, E, and f conversions, the maximum number of significant digits for the g and G conversions, or the maximum number of characters to be written from a string in s conversion. The precision takes the form of a period (.) followed either by an asterisk * (described later) or by an optional decimal integer; if only the period is specified, the precision is taken as zero. If a precision appears with any other conversion specifier, the behavior is undefined.
- An optional modifier h, l (ell), or L indicating the size of the receiving object. For instance, the conversion specifier d is preceded by h if the corresponding argument is a short int rather than an int (the argument will have been promoted according to the rules of integral promotions, and its value will be converted to short int before printing.) A table of modifiers is presented below, under the list of conversion specifiers.

As noted above, a field width, or precision, or both, may be indicated by an asterisk. In this case, an int argument supplies the field width or precision. The argument specifying field width, or precision, or both, should appear (in that order) before the argument (if any) to be converted. A negative field width argument is taken as a - flag followed by a positive field width. A negative precision argument is taken as if the precision were omitted.

The flag characters and their meanings are:

- The result of the conversion will be left-justified within the field. (It will be right-justified if this flag is not specified.)
- + The result of a signed conversion will always begin with a plus or minus sign. (It will begin with a sign only when a negative value is converted if this flag is not specified.)
- space If the first character of a signed conversion is not a sign, or if a signed conversion results in no characters, a space will be prefixed to the result. If the space and + flags both appear, the space flag will be ignored.

- # The result is converted to an “alternate form”. For `o` conversion, it increases the precision to force the first digit of the result to be a zero. For `x` (or `X`) conversion, a nonzero result will have `0x` (or `0X`) prefixed to it. For `e`, `E`, `f`, `g`, and `G` conversions, the result will always contain a decimal-point character, even if no digits follow it. (Normally, a decimal-point character appears in the result of these conversions only if a digit follows it.) For `g` and `G` conversions, trailing zeros will *not* be removed from the result. For other conversions, the behavior is undefined.
- 0 For `d`, `i`, `o`, `u`, `x`, `X`, `e`, `E`, `f`, `g`, and `G` conversions, leading zeros (following any indication of sign or base) are used to pad the field width; no space padding is performed. If the `0` and `-` flags both appear, the `0` flag will be ignored. For `d`, `i`, `o`, `u`, `x`, and `X` conversions, if a precision is specified, the `0` flag will be ignored. For other conversions, the behavior is undefined.

The conversion specifiers and their meanings are:

- `d` or `i` The argument is assumed to be an `int` and is converted to signed decimal notation. The precision specifies the minimum number of digits that appear.
- `o` The argument is assumed to be an unsigned `int` and is converted to unsigned octal notation. The precision specifies the minimum number of digits to appear.
- `u` The argument is assumed to be an unsigned `int` and is converted to unsigned decimal notation. The precision specifies the minimum number of digits to appear.
- `x` or `X` The argument is assumed to be an unsigned `int` and is converted to hexadecimal notation. The precision specifies the minimum number of digits to appear.
- `f` The argument is assumed to be a `double` and is converted to decimal notation in the form `[-]ddd.ddd`. The precision specifies the number of digits to appear after the decimal point. The default precision is six. If the precision is zero, no decimal-point is printed. The value is rounded to the appropriate number of digits.

e or E	The argument is assumed to be a double and is converted to decimal notation in the form [-]d.ddde(E)+dd. The precision specifies the number of digits to appear after the decimal-point. The default precision is six. If the precision is zero, no decimal-point is printed. The value is rounded to the appropriate number of digits.
g or G	The argument is assumed to be a double and is converted to decimal notation in the form of either e or f. This depends on the value of the converted number. The f form is used unless the exponent is less than -4 or greater than or equal to the precision. The precision specifies the number of significant digits. The decimal-point character appears only if a digit follows it.
c	The argument is assumed to be an int and is converted to an unsigned char.
s	The argument is assumed to be a pointer to a string. Characters in the string are printed until a null character is detected or until the number of characters indicated by the precision is exhausted.
p	The argument is assumed to be a pointer to a void and the value of the pointer is printed as a hexadecimal number.
n	The argument is assumed to be pointer to an integer into which is written the number of characters written by this call so far.
%	A % character is written.

Below is a table of modifiers that can precede a conversion specifier. If a modifier appears with any conversion specifier not listed, the behavior is undefined.

Modifier	Specifier	Default Argument Type	Modified Argument Type
h	d,i	int	short int
h	o,u,x,X	unsigned int	unsigned short
h	n	pointer to int	pointer to short int
l	d,i	int	long
l	o,u,x,X	unsigned int	unsigned long
l	n	pointer to int	pointer to long int
L	e,E,f,g,G	double	long double

... Arguments 1 through n are written by `fprintf()` according to the specifications contained in the format control string.

Return Value

This function returns the number of characters written. It returns a negative number if a write error occurs and sets `errno`.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

fputc

Writes a character to a stream.

```
#include <stdarg.h>
#include <stdio.h>
int fputc(
    int c,          /* character */
    FILE *stream    /* stream pointer */
)
```

Description

The `fputc()` function writes the character specified by `c` to the output stream pointed to by `stream` after converting it to an unsigned char. This function operates the same as the `putc` function.

If `stream` designates a disk file, its position indicator is advanced appropriately.

Arguments

<code>c</code>	Specifies the character to write.
<code>stream</code>	Points to an open pREPC+ output stream.

Return Value

This function returns the character written. If a write error occurs, the stream's error indicator is set, EOF is returned and `errno` is set.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

fgetc

fputs

Writes a string to a stream.

```
#include <stdarg.h>
#include <stdio.h>
int fputs(
    const char *s,    /* string */
    FILE *stream      /* stream pointer */
)
```

Description

The `fputs()` function writes the string pointed to by `s` to the stream pointed to by `stream`. The terminating null character is not written.

Arguments

<code>s</code>	Points to the string to be written.
<code>stream</code>	Points to an open pREPC+ stream.

Return Value

This function returns zero if the operation succeeds and EOF if it fails. If an error occurs, `errno` is set.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

`fgets`

fread

Reads from a stream.

```

#include <stdarg.h>
#include <stdio.h>
size_t fread(
    void *ptr,          /* buffer */
    size_t size,        /* element size */
    size_t nmemb,       /* element count */
    FILE *stream        /* stream pointer */
)

```

3

Description

This function reads up to `nmemb` elements whose size is specified by `size` from the stream pointed to by `stream` and puts them into the user buffer pointed to by `ptr`. The file position indicator for the stream (if defined) is advanced by the number of characters successfully read. If an error occurs, the resulting value of stream's file position indicator is indeterminate. If a partial item is read, its value is indeterminate.

Arguments

<code>ptr</code>	Points to the user buffer where items are stored.
<code>size</code>	Specifies the size of each item.
<code>nmemb</code>	Specifies the number of items to read.
<code>stream</code>	Points to an open pREPC+ stream.

Return Value

This function returns a count of the number of items successfully read. If an error occurs, stream's error indicator and `errno` are set. If `size` or `nmemb` is zero, `fread()` returns 0 and the contents of the array and the state of the stream remain unchanged.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

`fwrite`, `fopen`

free Deallocates memory.

```
#include <stdlib.h>
void free(
    void *ptr    /* pointer to memory segment */
)
```

Description

This function deallocates a specified memory segment (`ptr`). If the memory segment was not previously allocated by `calloc()`, `malloc()` or `realloc()`, or if the space has been deallocated by a call to `free()` or `realloc()`, the results are unpredictable.

Arguments

`ptr` Points to the memory segment to deallocate.

Return Value

This function does not return a value. If an error occurs, `errno` is set.

Error Codes

Refer to Appendix B.

Notes

`free()` calls the pSOS+ region manager to deallocate the memory.

Callable From

- Task

See Also

`calloc`, `malloc`, `realloc`

freopen

Reopens a file.

```
#include <stdarg.h>
#include <stdio.h>
FILE *freopen(
    const char *filename,    /* filename */
    const char *mode,        /* access mode */
    FILE *stream             /* stream pointer */
)
```

Description

The `freopen()` function first closes the file specified by `stream`. Then it opens the file named by the string pointed to by `filename` and associates it with the stream pointed to by `stream`. The file is opened in the mode specified by `mode`, which is interpreted just as in `fopen`. The error and end-of-file indicators for the stream are cleared.

If the close operation fails, `filename` is still opened and attached to `stream`.

This call can be used to rename `stdin`, `stdout`, and `stderr`.

Arguments

<code>filename</code>	Points to the name of the file to be opened.
<code>mode</code>	Points to the mode string, which specifies how to open the file.
<code>stream</code>	Points to an open pREPC+ stream.

Return Value

This function returns a file pointer if the file specified by `filename` is opened, or it returns a null pointer if it does not open the file. If an error occurs, `errno` is set.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

`fopen`, `fclose`

fscanf

Reads formatted input from a stream.

```
#include <stdarg.h>
#include <stdio.h>
int fscanf(
    FILE *stream,          /* stream pointer */
    const char *format,    /* format control string */
    ...                    /* arguments 1 through n */
)
```

Description

The `fscanf()` function reads input from the stream specified by `stream`. As input is read, it is divided into input fields. An input field is defined as a string of non-white space characters. It extends either to the next white space character or up to a specified field width. The input fields are handled in a manner determined by a format control string. The input fields are converted to data items and stored in variables pointed to by the remaining arguments. If insufficient arguments are provided for `format`, the behavior is undefined. If `format` is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

Arguments

<code>stream</code>	Points to an open pREPC+ stream.
<code>format</code>	<p>Points to the format control string. The format is a multibyte character sequence, beginning and ending with its initial shift state. It is composed of zero or more directives: one or more white-space characters; an ordinary multibyte character (neither <code>%</code> nor a white-space character); or a conversion specification. Each conversion specification is introduced by the character <code>%</code>. After the <code>%</code>, the following appear in sequence:</p> <ul style="list-style-type: none">■ An optional assignment-suppressing character <code>*</code>.■ An optional nonzero decimal integer that specifies the maximum field width.

- An optional modifier `h` or `l` (ell), indicating the size of the receiving object. For instance, the conversion specifier `d` is preceded by `h` if the corresponding argument is a pointer to a `short int` rather than a pointer to an `int`. A table of modifiers is presented below, under the list of conversion specifiers.
- A character that specifies the type of conversion to be applied. The valid conversion specifiers are described below.

`fscanf()` executes each directive of the format in turn. If a directive fails, as detailed below, `fscanf()` returns. Failures are described as input failures (due to the unavailability of input characters), or matching failures (due to inappropriate input).

A directive composed of white-space character(s) is executed by reading input up to the first non-white-space character (which remains unread), or until no more characters can be read.

A directive that is an ordinary multibyte character is executed by reading the next characters of the stream. If one of the characters differs from one comprising the directive, the directive fails, and the differing and subsequent characters remain unread.

A directive that is a conversion specification defines a set of matching input sequences, as described below for each specifier. A conversion specification is executed in the following steps:

Input white-space characters (as specified by the `isspace()` function) are skipped, unless the specification includes a `['`, `c`, or `n` specifier.

An input item is read from the stream, unless the specification includes an `n` specifier. An input item is defined as the longest matching sequence of input characters, unless that exceeds a specified field width, in which case it is the initial subsequence of that length in the sequence. The first character, if any, after the input item remains unread. If the length of the input item is zero, the execution of the directive fails: this condition is a matching failure, unless an error prevented input from the stream, in which case it is an input failure.

Except in the case of a `%` specifier, the input item (or, in the case of a `%n` directive, the count of input characters) is converted to a type appropriate to the conversion specifier. If the input item is not a matching sequence, the execution of the directive fails: this condition is a matching failure. Unless assignment suppression was indicated by a `*`, the result of the conversion is placed in the object pointed to by the first argument following the `format` argument that has not already received a conversion result. If this object does not have an appropriate type, or if the result of the conversion cannot be represented in the space provided, the behavior is undefined.

The conversion specifier characters and their definitions are as follows:

- `d, i` An optionally signed decimal integer is expected. The corresponding argument should be a pointer to an integer.
- `o` An optionally signed octal integer is expected. The corresponding argument should be a pointer to an integer.
- `u` An optionally signed octal integer is expected. The corresponding argument should be a pointer to an unsigned long integer.
- `x` An optionally signed hexadecimal integer is expected. The corresponding argument should be a pointer to an integer.
- `e, f, g` An optionally signed floating-point number is expected. The corresponding argument should be a pointer to a float.
- `p` An unsigned hexadecimal number is expected. The corresponding argument should be a pointer to a pointer to void.
- `s` A sequence of non-white-space characters is expected. The corresponding argument should be a pointer to a character array large enough to accept the sequence and an added, terminating null character.
- `c` A sequence of characters is expected. The number of characters in the sequence should be equal to the field width. If the field width is not specified, one character is expected. The corresponding argument should be a pointer to a character array large enough to accept the sequence. A terminating null character is not added.

- [A sequence of characters is expected. Every character must match one of the characters listed after the [character and up to and including a] character. If the first character listed after the initial [character is a circumflex (^) character, then the characters read must not match the characters given in the list. A character list beginning with [] or [^] is a special case. If this occurs, the first] character does not end the list and a second] character is needed. The corresponding argument should be a pointer to a character array large enough to accept the sequence and an added, terminating null character.
- n No input is read. The corresponding argument should be a pointer to an integer that is loaded with the number of characters read so far.
- % A % character is expected. No assignment occurs.

Below is a table of the modifiers that can precede a conversion specifier. If a modifier appears with any conversion specifier not listed, the behavior is undefined.

Modifier	Specifier	Default Argument Type	Modified Argument Type
h	d,i,n	pointer to int	pointer to short
h	o,x,u	pointer to unsigned int	pointer to unsigned short
l	d,i,n	pointer to int	long
l	o,x,u	pointer to unsigned int	pointer to unsigned long
l	e,f,g	pointer to float	pointer to double

The L modifier is not supported by fscanf().

... Arguments 1 through n point to variables where input is stored.

Return Value

This function returns EOF and sets errno if an input failure occurs before any conversion. Otherwise, it returns the number of input items assigned, which can be fewer than provided, even zero, in the event of an early matching failure.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

scanf

fseek Sets the file position indicator.

```
#include <stdarg.h>
#include <stdio.h>
int fseek(
    FILE *stream,    /* stream pointer */
    long offset,     /* file offset */
    int whence       /* relative file base */
)
```

3

Description

The `fseek()` function sets the file position indicator for the stream pointed to by `stream`.

For a text stream, either `offset` should be zero, or `offset` should be a value returned by an earlier call to `ftell` function on the same stream and `base` should be `SEEK_SET`.

A successful call to `fseek()` clears the stream's end-of file indicator and undoes any effect of `ungetc` function on the same stream.

If `stream` refers to an I/O device, this function does nothing and returns a zero.

Arguments

<code>stream</code>	Points to an open pREPC+ stream.
<code>offset</code>	For a binary stream, specifies the offset value, which will be added to the position specified by <code>whence</code> to calculate the new value of the position indicator.
<code>whence</code>	Specifies the relative file base. <code>whence</code> can have one of the following values:

Value	Description
<code>SEEK_SET</code>	Beginning of file
<code>SEEK_CUR</code>	Current position in file
<code>SEEK_END</code>	End of file

Return Value

This function returns zero if the operation is successful or a nonzero number if unsuccessful. If an error occurs, `errno` is set.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

`ftell`, `fsetpos`, `fgetpos`

fsetpos Sets file position by using the fgetpos result.

```
#include <stdarg.h>
#include <stdio.h>
int fsetpos(
    FILE *stream,      /* stream pointer */
    const fpos_t *pos  /* stream position */
)
```

3

Description

The `fsetpos()` function sets the position indicator for the stream pointed to by `stream` to the value of the object pointed to by `pos`.

A successful call to `fsetpos` function clears the EOF indicator for the stream and undoes any effects of the `ungetc` function on the same stream.

Arguments

<code>stream</code>	Points to an open pREPC+ stream. If <code>stream</code> refers to an I/O device, this function does nothing and returns a 0.
<code>pos</code>	Points to an object that specifies the new value of the position indicator. The object should contain a value previously returned by the <code>fgetpos()</code> function on the same stream.

Return Value

This function returns a 0 if successful and a nonzero number if unsuccessful. If an error occurs, `errno` is set.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

fgetpos

ftell Gets the file position indicator.

```
#include <stdarg.h>
#include <stdio.h>
long ftell(
    FILE *stream    /* stream pointer */
)
```

3

Description

The `ftell()` function obtains the current value of the position indicator for the stream pointed to by `stream`. This value can be passed to `fseek()` as an input parameter.

For a binary stream, the position indicator is the number of characters from the beginning of the file. For a text stream, the position indicator contains unspecified information, usable by `fseek` function.

If `stream` refers to an I/O device, this function does nothing and returns a zero.

Arguments

`stream` Points to an open pREPC+ stream.

Return Value

If successful, this function returns the current file position indicator. It returns EOF if an error occurs, and sets the `errno`.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

`fseek`, `fsetpos`, `fgetpos`

fwrite

Writes to a stream.

```
#include <stdarg.h>
#include <stdio.h>
size_t fwrite(
    const void *ptr,      /* output buffer */
    size_t size,          /* item size */
    size_t nmemb,         /* item count */
    FILE *stream           /* stream pointer */
)
```

3

Description

The `fwrite()` function writes, from the array pointed to by `ptr`, up to `nmemb` elements whose size is specified by `size` to the stream pointed to by `stream`.

The file position indicator for the stream (if defined) is advanced by the number of characters successfully written. If an error occurs, the resulting value of the file position indicator for the stream is indeterminate.

Arguments

<code>ptr</code>	Points to the output buffer.
<code>size</code>	Specifies the size of each item to write.
<code>nmemb</code>	Specifies the number of items to write.
<code>stream</code>	Points to an open pREPC+ stream.

Return Value

This function returns the number of items written. If an error occurs, `errno` is set.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

fread

getc

Gets a character from a stream.

```
#include <stdarg.h>
#include <stdio.h>
int getc(
    FILE *stream    /* stream pointer */
)
```

3

Description

This function is equivalent to the `fgetc()` function. It reads the next character from a specified stream.

Arguments

`stream` Points to an open pREPC+ stream.

Return Value

This function returns the character that is read. If an end-of-file condition is detected, the stream's end-of-file indicator is set. If a read error occurs, the stream's error flag is set. In both cases, `EOF` is returned. If an error occurs, `errno` is set.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

`putc`

getchar

Gets a character from `stdin`.

```
#include <stdarg.h>
#include <stdio.h>
int getchar(
    void
)
```

Description

The `getchar()` function reads the next character from the standard input device. It is equivalent to `getc(stdin)`.

Return Value

This function returns the character that is read. If EOF is detected, the `stdin` EOF flag is set. If a read error occurs, `stdin`'s error flag is set. In both cases, EOF is returned. If an error occurs, `errno` is set.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

`putchar`, `getc`

gets Gets a string from `stdin`.

```
#include <stdarg.h>
#include <stdio.h>
char *gets(
    char *s    /* buffer */
)
```

3

Description

The `gets()` function reads characters from the standard input device into a user buffer(`s`). It continues to read characters until a new-line character is read or an end-of-file condition is detected. Any new-line character is discarded, and a null character is added after the last character read into the user buffer.

Arguments

`s` Points to the user buffer.

Return Value

If successful, the `gets()` function returns `s`. If a read error occurs or an end-of-file condition is detected before any characters are read, a null pointer is returned. If an error occurs, `errno` is set.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

`puts`, `fgets`

gmtime

Converts the calendar time to broken-down time.

```
#include <time.h>
struct tm *gmtime (
    const time_t *timer    /* calendar time */
)
```

Description

The `gmtime()` function converts the calendar time pointed to by `timer` into broken-down time, expressed as Coordinated Universal Time (UTC).

The calendar time is generally obtained through a call to `time()`.

Arguments

`timer` Points to the calendar time.

Return Value

The `gmtime()` function always returns `NULL`, since the concept of UTC is not supported by pREPC+.

Error Codes

None.

Notes

Callable From

- Task

See Also

`gmtime_r`, `time`, `localtime`, `localtime_r`, `mktime`

gmtime_r (Reentrant) Converts the calendar time to broken-down time.

```
#include <time.h>
struct tm *gmtime_r (
    const time_t *timer,      /* calendar time */
    struct tm *resultp        /* result */
)
```

3

Description

`gmtime_r()` is the reentrant version of the ANSI function `gmtime()`, as defined by POSIX 1003.1c. It converts the calendar time pointed to by `timer` into broken-down time, expressed as Coordinated Universal Time (UTC). The broken-down time is stored in the structure pointed to by `resultp`.

The calendar time is generally obtained through a call to `time()`.

Arguments

<code>timer</code>	Points to the calendar time.
<code>resultp</code>	Points to the structure of type <code>tm</code> where <code>gmtime_r()</code> stores the result. The <code>tm</code> structure is defined in the <code>mktime()</code> description on page 3-111.

Return Value

`gmtime_r()` always returns `NULL`, since the concept of UTC is not supported by pREPC+.

Error Codes

No error codes are returned.

Notes

Callable From

- Task
- ISR

See Also

`gmtime`, `time`, `localtime`, `localtime_r`, `mktime`

isalnum Tests for an alphanumeric character.

```
#include <ctype.h>
int isalnum(
    int c    /* character */
)
```

Description

This function tests the value in `c` (converted to an unsigned char) to see if it is an alphanumeric character. An alphanumeric character is a character for which either `isdigit` or `isalpha` is true.

Arguments

`c` Specifies the value to be tested.

Return Value

This function returns a nonzero value if the test result is true. It returns a 0 if the result is false.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task
- ISR

See Also

`isalpha`, `isdigit`

isalpha Tests for an alphabetic character.

```
#include <ctype.h>
int isalpha(
    int c    /* character */
)
```

Description

This function tests the value in `c` (converted to an unsigned char) to see if it is an uppercase or lowercase letter.

Arguments

`c` Specifies the value to be tested.

Return Value

This function returns a nonzero value if the test result is true. It returns a 0 if the result is false.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task
- ISR

isctrl Tests for a control character.

```
#include <ctype.h>
int isctrl(
    int c    /* character */
)
```

Description

This function tests the value in `c` (converted to an unsigned char) to see if it is an ASCII control character. ASCII control characters are those whose values lie between 0 and 31, inclusive, and the character 127 (DEL).

Arguments

`c` Specifies the value to be tested.

Return Value

This function returns a nonzero value if the test result is true. It returns a 0 if the result is false.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task
- ISR

See Also

`isprint`

isdigit Tests for a digit.

```
#include <ctype.h>
int isdigit(
    int c    /* character */
)
```

Description

This function tests the value in `c` (converted to an unsigned char) to see if it is a decimal digit (0 through 9).

Arguments

`c` Specifies the value to be tested.

Return Value

This function returns a nonzero value if the test result is true. It returns a 0 if the result is false.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task
- ISR

isgraph Tests for a graphical character.

```
#include <ctype.h>
int isgraph(
    int c    /* character */
)
```

Description

This function tests the value in `c` (converted to an unsigned char) to see if it is an ASCII graphical character. ASCII graphical characters are those whose values lie from 33 (exclamation point) through 126 (tilde). This includes all of the printable characters except the space (' ').

Arguments

`c` Specifies the value to be tested.

Return Value

This function returns a nonzero value if the test result is true. It returns a 0 if the result is false.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task
- ISR

See Also

`isprint`

islower Tests for a lowercase letter.

```
#include <ctype.h>
int islower(
    int c    /* character */
)
```

Description

This function tests the value in `c` (converted to an unsigned char) to see if it is a lowercase letter.

Arguments

`c` Specifies the value to be tested.

Return Value

This function returns a nonzero value if the test result is true. It returns a 0 if the result is false.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task
- ISR

See Also

`isupper`, `isalpha`

isprint Tests for a printable character.

```
#include <ctype.h>
int isprint(
    int c    /* character */
)
```

Description

This function tests the value in `c` (converted to an unsigned char) to see if it is an ASCII printable character. An ASCII printable character is any character that is not a control character. Their values lie between 32 (space) and 126 (tilde), inclusive.

Arguments

`c` Specifies the value to be tested.

Return Value

This function returns a nonzero value if the test result is true. It returns a 0 if the result is false.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task
- ISR

See Also

`isgraph`, `iscntrl`

ispunct Tests for a punctuation character.

```
#include <ctype.h>
int ispunct(
    int c    /* character */
)
```

Description

This function tests the value in `c` (converted to an unsigned char) to see if it is an ASCII punctuation mark character. An ASCII punctuation mark character is any printing character that is neither a space nor a character for which `isalnum` is true.

Arguments

`c` Specifies the value to be tested.

Return Value

This function returns a nonzero value if the test result is true. It returns a 0 if the result is false.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task
- ISR

See Also

`isalnum`, `isprint`, `isalpha`, `isdigit`

isspace Tests for a space.

```
#include <ctype.h>
int isspace(
    int c    /* character */
)
```

Description

This function tests the value in `c` (converted to an unsigned char) to see if it is a tab (`'\t'`), line-feed (`'\n'`), vertical tab (`'\v'`), form-feed (`'\f'`), carriage return (`'\r'`), or space character (`' '`).

Arguments

`c` Specifies the value to be tested.

Return Value

This function returns a nonzero value if the test result is true. It returns a 0 if the result is false.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task
- ISR

isupper Tests for an uppercase letter.

```
#include <ctype.h>
int isupper(
    int c    /* character */
)
```

Description

This function tests the value in `c` (converted to an unsigned char) to see if it is an uppercase letter.

Arguments

`c` Specifies the value to be tested.

Return Value

This function returns a nonzero value if the test result is true. It returns a 0 if the result is false.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task
- ISR

See Also

`isalpha`, `islower`

isxdigit Tests for a hexadecimal digit.

```
#include <ctype.h>
int isxdigit(
    int c    /* character */
)
```

Description

This function tests the value in `c` (converted to an unsigned char) to see if it is a hexadecimal digit. The hexadecimal digits are defined as the ASCII representation of digits 0 through 9, lower case letters a through f and uppercase letters A through F.

Arguments

`c` Specifies the value to be tested.

Return Value

This function returns a nonzero value if the test result is true. It returns a 0 if the result is false.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task
- ISR

See Also

`isdigit`

labs Computes the absolute value of a long integer.

```
#include <stdlib.h>
long labs (
    long j    /* long integer */
)
```

Description

The `labs()` function converts the long integer `j` into its absolute value. If the result cannot be represented, the behavior is undefined.

Arguments

`j` Specifies the long integer to be converted.

Return Value

`labs()` returns the absolute value.

Error Codes

None.

Notes

Callable From

- Task
- ISR

See Also

`labs`

ldiv Performs a division operation on two specified long integers.

```
#include <stdlib.h>
ldiv_t ldiv (
    long numer,      /* numerator */
    long denom       /* denominator */
)
```

3

Description

The `ldiv()` function computes the quotient and remainder of the division of the numerator `numer` by the denominator `denom`. If the division is inexact, the resulting quotient is the integer of lesser magnitude that is the nearest to the algebraic quotient. If the result cannot be represented, the behavior is undefined; otherwise, `quot * denom + rem` is equal to `numer`.

Arguments

<code>numer</code>	Specifies the numerator.
<code>denom</code>	Specifies the denominator.

Return Value

The `ldiv()` function returns a structure of type `ldiv_t`. This structure is defined in `stdlib.h` as follows:

```
typedef struct {
    long quot; /* the quotient */
    long rem;  /* the remainder */
} ldiv_t;
```

Error Codes

None.

Notes

Callable From

- Task
- ISR

See Also

`div`

localeconv Obtains the current locale settings.

```
#include <locale.h>
struct lconv *localeconv(void)
```

Description

The `localeconv()` function obtains the current locale settings that relate to numeric values, putting them into a statically allocated structure of type `lconv`. It returns a pointer to that structure.

The members of `lconv` with type `char *` are pointers to strings, any of which (except `decimal_point`) can point to `" "`, to indicate that the value is not available in the current locale or is of zero length. The members with type `char` are nonnegative numbers, any of which can be `CHAR_MAX` to indicate the value is not available in the current locale.

The `lconv` structure is defined as follows:

```
struct lconv {
    char *decimal_point;    /* Decimal point character for
                           * non-monetary values */
    char *thousands_sep;   /* Thousands separator for
                           * non-monetary values */
    char *grouping;         /* Specifies grouping for
                           * non-monetary values */
    char *int_curr_symbol;  /* International currency symbol */
    char *currency_symbol; /* The local currency symbol */
    char *mon_decimal_point; /* Decimal point character for
                           * monetary values */
    char *mon_thousands_sep; /* Thousands separator for
                           * monetary values */
    char *mon_grouping;     /* Specifies grouping for
                           * monetary values */
    char *positive_sign;    /* Positive value indicator
                           * for monetary values */
    char *negative_sign;    /* Negative value indicator
                           * for monetary values */
    char int_frac_digits;   /* Number of digits displayed
                           * to the right of the decimal
                           * point for monetary values
                           * displayed using international
                           * format */
    char frac_digits;       /* Number of digits displayed
                           * to the right of the decimal
                           * point for monetary values */
}
```

```

char p_cs_precedes; /* 1 if currency symbol precedes
                    * positive value, 0 if currency
                    * symbol follows value */
char p_sep_by_space; /* 1 if currency symbol is
                    * separated from value by a
                    * space, 0 otherwise */
char n_cs_precedes; /* 1 if currency symbol precedes
                    * a negative value, 0 if currency
                    * symbol follows value */
char n_sep_by_space; /* 1 if currency symbol is
                    * separated from a negative value
                    * by a space, 0 if currency symbol
                    * follows value */
char p_sign_posn; /* Indicates position of positive
                  * value symbol */
char n_sign_posn; /* Indicates position of negative
                  * value symbol */
};

```

The elements of grouping and mon_grouping are interpreted according to the following:

CHAR_MAX	No further grouping is to be performed.
0	The previous element is to be repeatedly used for the remainder of the digits.
<i>other</i>	The integer value is the number of digits that comprise the current group. The next element is examined to determine the size of the next group of digits before the current group.

The value of p_sign_posn and n_sign_posn is interpreted according to the following:

0	Parentheses surround the quantity and currency_symbol.
1	The sign string precedes the quantity and currency_symbol.
2	The sign string succeeds the quantity and currency_symbol.
3	The sign string immediately precedes currency_symbol.
4	The sign string immediately precedes currency_symbol.

Return Value

This function returns a pointer to the filled-in lconv structure. This structure must not be changed by your program, but it may be overwritten by a subsequent call to

`localeconv()`. In addition, calls to `setlocale()` with categories `LC_ALL`, `LC_MONETARY`, or `LC_NUMERIC` may overwrite the contents of the structure.

Error Codes

None.

Notes

Callable From

- Task
- ISR

See Also

`setlocale`

localtime

Converts the calendar time to broken-down time.

```
#include <time.h>
struct tm *localtime (
    const time_t *timer    /* calendar time */
)
```

Description

The `localtime()` function converts the calendar time pointed to by `timer` into broken-down time. The time is represented in local time.

The calendar time is generally obtained through a call to `time()`.

Arguments

`timer` Points to the calendar time.

Return Value

The `localtime()` function returns a pointer to the `tm` structure that contains the broken-down time. The `tm` structure is defined in the `mktime()` description on page 3-111.

Error Codes

None.

Notes

Callable From

- Task

See Also

`localtime_r`, `time`, `gmtime`, `gmtime_r`, `mktime`

localtime_r (Reentrant) Converts the calendar time to broken-down time.

```
#include <time.h>
struct tm *localtime_r (
    const time_t *timer,      /* pointer to calendar time */
    struct tm *resultp        /* pointer to result */
)
```

3

Description

`localtime_r()` is the reentrant version of the ANSI function `localtime()`, as defined by POSIX 1003.1c. It converts the calendar time pointed to by `timer` into broken-down time and stores it in the structure pointed to by `resultp`. The time is represented in local time.

The calendar time is generally obtained through a call to `time()`.

Arguments

<code>timer</code>	Points to the calendar time.
<code>resultp</code>	Points to the <code>tm</code> structure where <code>localtime_r()</code> stores the result. The <code>tm</code> structure is defined in the <code>mktime()</code> description on page 3-111.

Return Value

Upon success, `localtime_r()` returns the value of `resultp`. On failure, it returns `NULL`; the only cause of failure is if `timer` points to a negative value.

Error Codes

No error codes are returned.

Notes

Callable From

- Task
- ISR

See Also

`localtime`, `time`, `gmtime`, `gmtime_r`, `mktime`

malloc Allocates memory.

```
#include <stdlib.h>
void *malloc(
    size_t size    /* element size */
)
```

Description

The `malloc()` function allocates memory for an object whose size is specified by `size` and whose value is indeterminate. `malloc()` calls the pSOS+ region manager to allocate the memory. The caller can be blocked if memory is not available and the wait option is selected in the pREPC+ Configuration Table. Memory is allocated from Region 0.

Arguments

`size` Specifies the number of bytes to allocate.

Return Value

This function returns either a pointer to the allocated memory or a null pointer if no memory is allocated. If an error occurs, `errno` is set.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

mblen

Determines the number of bytes in a multibyte character.

```
#include <stdlib.h>
int mblen (
    const char *s,      /* character */
    size_t n           /* size of character */
)
```

Description

If *s* is not a null pointer, the `mblen()` function determines the number of bytes contained in the multibyte character pointed to by *s*. Except that the shift state of the `mbtowc()` function is not affected, it is equivalent to:

```
mbtowc((wchar_t *)0, s, n);
```

Arguments

<i>s</i>	Points to the character to be examined.
<i>n</i>	Specifies the size of <i>s</i> .

Return Value

If *s* is a null pointer, this function returns a nonzero or zero value, if multibyte character encodings, respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, this function either returns 0 (if *s* points to the null character), or returns the number of bytes that are contained in the multibyte character (if the next *n* or fewer bytes form a valid multibyte character), or returns -1 (if they do not form a valid multibyte character).

Error Codes

None.

Notes

Callable From

- Task
- ISR

See Also

`mbtowc`

mbstowcs Converts a multibyte character string into a wide character string.

```
#include <stdlib.h>
size_t mbstowcs (
    wchar_t *pwcs,    /* wide string */
    const char *s,    /* original multibyte string */
    size_t n          /* wide string length */
)
```

Description

The `mbstowcs()` function converts a sequence of multibyte characters that begins in the initial shift state from the array pointed to by `s` into a sequence of corresponding codes and stores not more than `n` codes into the array pointed to by `pwcs`. No multibyte characters that follow a null character (which is converted into a code with value zero) will be examined or converted. Each multibyte character is converted as if by a call to the `mbtowc()` function, except that the shift state of the `mbtowc()` function is not affected.

If copying takes place between objects that overlap, the behavior is undefined.

Arguments

<code>pwcs</code>	Points to the array where <code>mbstowcs()</code> stores the converted string.
<code>s</code>	Points to the string to be converted.
<code>n</code>	Specifies the length of <code>pwcs</code> .

Return Value

If an invalid multibyte character is encountered, `mbstowcs()` returns `(size_t)-1`. Otherwise, it returns the number of array elements modified, not including a terminating zero code, if any.

Error Codes

None.

Notes

Callable From

- Task
- ISR

See Also

`mbtowc`, `wctombs`, `wctomb`

mbtowlc Converts a multibyte character into its wide character equivalent.

```
#include <stdlib.h>
int mbtowlc (
    wchar_t *pwc,      /* result wide character */
    const char *s,      /* original multibyte character */
    size_t n           /* size of original character */
)
```

Description

If *s* is not a null pointer, the `mbtowlc()` function determines the number of bytes that are contained in the multibyte character pointed to by *s*. It then determines the code for the value of type `wchar_t` that corresponds to that multibyte character. (The value of the code corresponding to the null character is zero.) If the multibyte character is valid and *pwc* is not a null pointer, the `mbtowlc()` function stores the code in the object pointed to by *pwc*. At most *n* bytes of the array pointed to by *s* will be examined.

Arguments

<i>pwc</i>	Points to the array where <code>mbtowlc()</code> stores the result character.
<i>s</i>	Points to the multibyte character to be converted.
<i>n</i>	Specifies the size of the multibyte character to be converted.

Return Value

If *s* is a null pointer, this function returns a nonzero or zero value, if multibyte codings, respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, this function either returns 0 (if *s* points to the null character), or returns the number of bytes that are contained in the converted multibyte character (if the next *n* or fewer bytes form a valid multibyte character), or returns -1 (if they do not form a valid multibyte character).

In no case will the value returned be greater than *n* or the value of the `MB_CUR_MAX` macro.

Error Codes

None.

Notes

Callable From

- Task
- ISR

See Also

mbstowcs, wctomb, wcstombs

memchr

Searches memory for a character.

```
#include <string.h>
void *memchr(
    const void *s,    /* target buffer */
    int c,            /* character key */
    size_t n          /* search length */
)
```

Description

This function searches for the first occurrence of the character `c` (converted to an unsigned char) in the first `n` characters of the object pointed to by `s`.

Arguments

<code>s</code>	Points to the buffer to be searched.
<code>c</code>	Specifies the character to be searched for.
<code>n</code>	Specifies the number of characters to search through.

Return Value

This function returns a pointer to the located character, or a null pointer if the character is not found.

Error Codes

None.

Notes

Callable From

- Task
- ISR

See Also

`strchr`

memcmp

Compares two objects in memory.

```
#include <string.h>
int memcmp(
    const void *s1,    /* buffer 1 */
    const void *s2,    /* buffer 2 */
    size_t n           /* comparison length */
)
```

Description

This function compares *n* characters in the buffers pointed to by *s1* and *s2*.

Arguments

<i>s1</i>	Points to the first buffer.
<i>s2</i>	Points to the second buffer.
<i>n</i>	Specifies the number of characters to be compared.

Return Value

This function returns a value that is either greater than, equal to, or less than zero. The result depends on whether the object pointed to by *s1* is greater than, equal to, or less than the object pointed to by *s2*.

Error Codes

None.

Notes

Callable From

- Task
- ISR

See Also

`strcmp`

memcpy

Copies characters in memory.

```
#include <string.h>
void *memcpy(
    void *s1,          /* destination address */
    const void *s2,     /* source address */
    size_t n           /* source length */
)
```

Description

This function copies `n` characters from the object pointed to by `s2` into the object pointed to by `s1`. If the memory areas overlap, the result is unpredictable.

Arguments

<code>s1</code>	Points to the source object.
<code>s2</code>	Points to the destination object.
<code>n</code>	Specifies the number of characters to be copied.

Return Value

This function returns the value of `s1`.

Error Codes

None.

Notes

Callable From

- Task
- ISR

See Also

`memmove`, `strcpy`, `strncpy`

memmove Copies characters in memory.

```
#include <string.h>
void *memmove(
    void *s1,           /* destination address */
    const void *s2,     /* source address */
    size_t n            /* source length */
)
```

Description

This function copies a specified number of characters from one object into another. `memmove()` copies the data correctly even if the memory areas overlap (unlike `memcpy()`).

Arguments

<code>s1</code>	Points to the source object.
<code>s2</code>	Points to the destination object.
<code>n</code>	Specifies the number of characters to be copied.

Return Value

This function returns the value of `s1`.

Error Codes

None.

Notes

Callable From

- Task
- ISR

See Also

`memcpy`, `strcpy`, `strncpy`

memset

Initializes memory with a given value.

```
#include <string.h>
void *memset(
    void *s,    /* object address */
    int c,      /* initialization value */
    size_t n    /* number of characters */
)
```

Description

The `memset()` function copies the value of `c` (converted to an unsigned char) into each of the first `n` characters in the object pointed to by `s`.

Arguments

<code>s</code>	Points to the object where the character is to be copied.
<code>c</code>	Specifies the value to be copied.
<code>n</code>	Specifies the number of characters in the object to be initialized.

Return Value

The function returns the value of `s`.

Error Codes

None.

Notes

Callable From

- Task
- ISR

mktime

Converts the broken-down time into calendar time.

```
#include <time.h>
time_t mktime (
    struct tm *timeptr    /* broken-down time */
)
```

Description

The `mktime()` function converts the broken-down time, expressed as local time, in the structure pointed to by `timeptr` into calendar time with the same encoding as the `time()` function. This function is primarily used to initialize the system time. The elements `tm_wday` and `tm_yday` are set by the function, so they need not be defined prior to the call. The original values of the other components are not restricted to the normal ranges (see below).

Upon successful completion, the values of `tm_wday` and `tm_yday` are set appropriately, and the other components are set to represent the specified calendar time, but with their values forced to the normal ranges; the final value of `tm_mday` is not set until `tm_mon` and `tm_year` are determined.

Arguments

`timeptr` Points to a structure of type `tm` that stores the broken-down time. The `tm` structure is defined as follows. The semantics of the members and their normal ranges are expressed in the comments.

```
struct tm {
    int tm_sec;    /* seconds after the minute [0,61] */
    int tm_min;    /* minutes after the hour [0,59] */
    int tm_hour;   /* hours since midnight [0,23] */
    int tm_mday;   /* day of the month [1,31] */
    int tm_mon;    /* months since January [0,11] */
    int tm_year;   /* years since 1900 */
    int tm_wday;   /* days since Sunday [0,6] */
    int tm_yday;   /* days since January 1 [0,365] */
    int tm_isdst;  /* Daylight Saving Time flag */
};
```

The range [0, 61] for `tm_sec` allows for up to two leap seconds. The value of `tm_isdst` is positive if Daylight Saving Time is in effect, zero if Daylight Saving Time is not in effect, and negative if the information is not available.

Return Value

The `mktime()` function returns the specified calendar time encoded as a value of type `time_t`. If the calendar time cannot be represented, the function returns the value `(time_t) -1`.

Error Codes

None.

Example

What day of the week is July 4, 2001?

```
#include <stdio.h>
#include <time.h>
static const char *const wday[] = {
    "Sunday", "Monday", "Tuesday", "Wednesday",
    "Thursday", "Friday", "Saturday", "-unknown-"
};
struct tm time_str;

/* ... */

time_str.tm_year = 2001 - 1900;
time_str.tm_mon = 7 - 1;
time_str.tm_mday = 4;
time_str.tm_hour = 0;
time_str.tm_min = 0;
time_str.tm_sec = 1;
time_str.tm_isdst = -1;

if (mktime(&time_str) == -1)
    time_str.tm_wday = 7;

printf("%s\n", wday[time_str.tm_wday]);
```

Notes

Callable From

- Task
- ISR

See Also

`time`, `localtime`, `localtime_r`

perror Prints a diagnostic message.

```
#include <stdarg.h>
#include <stdio.h>
void perror(
    const char *s    /* error string */
)
```

Description

The `perror()` function writes the string pointed to by `s` followed by a diagnostic message to the standard error device. The diagnostic message is a function of the calling task's `errno` value.

Arguments

`s` Points to the string to write (error message).

Return Value

This function does not return a value.

Error Codes

No error codes are returned.

Notes

Callable From

- Task

See Also

`errno`

printf Prints formatted output to `stdout`.

```
#include <stdarg.h>
#include <stdio.h>
long printf(
    const char *format,    /* format control */
    ...                  /* arguments 1 through n */
)
```

3

Description

The `printf()` function is equivalent to `fprintf()`, except that the output is directed to the standard output device instead of a file designated by an input parameter.

Arguments

<code>format</code>	Points to the format control string. For more information, see <code>fprintf</code> on page 3-43.
<code>...</code>	Arguments 1 through <code>n</code> to be written according the specifications of the control string.

Return Value

This function returns either the number of characters written or `EOF` if a write error occurs. If an error occurs, `errno` is set.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

`scanf`, `fprintf`

putc Writes a character to a stream.

```
#include <stdarg.h>
#include <stdio.h>
int putc(
    int c,          /* character */
    FILE *stream    /* stream pointer */
)
```

3

Description

The `putc()` function writes the character specified by `c` (converted to an unsigned char) to the stream pointed to by `stream`. `stream`'s position indicator (if defined) is advanced on a successful write.

Arguments

<code>c</code>	Specifies the character to write.
<code>stream</code>	Points to an open pREPC+ stream.

Return Value

The `putc()` function returns `c`. If a write error occurs, the error flag for the stream is set, `EOF` is returned, and `errno` is set.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

`getc`, `putchar`

putchar

Writes a character to stdout.

```
#include <stdarg.h>
#include <stdio.h>
int putchar(
    int c    /* character */
)
```

Description

The `putchar()` function writes the character `c` to the standard output stream after converting it to an unsigned char.

Arguments

`c` Specifies the character to write.

Return Value

The `putchar()` function returns `letter`. If a write error occurs, the error flag for the standard output stream is set, `EOF` is returned, and `errno` is set.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

`getchar`, `putc`

puts Writes a string to a stream.

```
#include <stdarg.h>
#include <stdio.h>
int puts(
    const char *s    /* string */
)
```

3

Description

The `puts()` function writes a string to the standard output stream, and appends a new-line character to the output. The terminating null character is not written.

Arguments

`s` Points to the string to write.

Return Value

The `puts()` function returns 0 if the operation is successful and EOF if the operation fails. If an error occurs, the stream's error indicator and `errno` are set.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

`gets`, `fputs`

qsort

Sorts an array.

```
#include <stdlib.h>
void qsort(
    void *base,      /* array base */
    size_t nmemb,    /* array length */
    size_t size,     /* array element size */
    int (*compar) (const void *, const void *)
    /* comparison function */
)
```

Description

The `qsort()` function sorts an array of `nmemb` objects, the initial element of which is pointed to by `base`. The size of each object is specified by `size`.

The array is sorted in ascending order according to the user-supplied function pointed to by `compar`. The `compar` function is called with two arguments that point to the objects being compared. The `compar` function must return an integer that is less than, equal to, or greater than 0 if the first argument is considered less than, equal to, or greater than the second argument, respectively. The `compar` function can call all pREPC+ character handling functions and all pREPC+ string handling functions except `strtok()`. Other pREPC+ functions cannot be called from `compar`.

If two members of the array are equal, their order in the sorted array is unspecified.

Arguments

<code>base</code>	Points to the beginning of the array.
<code>nmemb</code>	Specifies the length of the array.
<code>size</code>	Specifies the size of each member of the array.
<code>compar</code>	Points to the user-supplied comparison function.

Return Value

This function does not return a value.

Error Codes

None.

Notes

Callable From

- Task
- ISR, provided the `compar` function can be called from an ISR.

See Also

`bsearch`

rand Returns a pseudo-random number.

```
#include <stdlib.h>
int rand(
    void
)
```

Description

The `rand()` function generates a pseudo-random integer number in the range 0 through 32,767. This function works in concert with `srand()`.

Return Value

This function returns the random number generated.

Error Codes

None.

Notes

Callable From

- Task

See Also

`srand`

realloc Allocates memory.

```
#include <stdlib.h>
void *realloc(
    void *ptr,      /* pointer */
    size_t size     /* new size */
)
```

3

Description

The `realloc()` function changes the size of the object pointed to by `ptr` to the size specified by `size`. The contents of the object remain unchanged up to the lesser of the new or old sizes. If the new size is larger, the value of the newly allocated portion of the object is indeterminate.

The caller can be blocked if memory is not available and the wait option is selected in the pREPC+ Configuration Table.

Arguments

<code>ptr</code>	Points to the object whose size is to be changed. If <code>ptr</code> is a null pointer, <code>realloc()</code> behaves like <code>malloc()</code> . If <code>ptr</code> does not match a segment previously returned by <code>calloc()</code> , <code>malloc()</code> or <code>realloc()</code> , the result is unpredictable. If <code>ptr</code> is not a null pointer and <code>size</code> is 0, the segment is deallocated.
<code>size</code>	Specifies the new size of the object.

Return Value

This function returns a pointer to the possibly moved allocated memory or a null pointer. If an error occurs, `errno` is set.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

`calloc`, `malloc`, `realloc`

remove	Removes a file.
---------------	-----------------

```
#include <stdarg.h>
#include <stdio.h>
int remove(
    const char *filename    /* file name */
)
```

Description

This function removes the file whose name is pointed to by `filename`. Before a file can be removed, it must be closed. This restriction does not apply to files residing on NFS volumes.

Arguments

filename	Points to the string that contains the filename.
----------	--

Return Value

This function returns zero if the file is removed. If the file is not removed or if `filename` specifies an I/O device, then a nonzero value is returned and `errno` is set.

Error Codes

Refer to Appendix B.

Notes

Callable From

- ## ■ Task

rename

Renames a file.

```
#include <stdarg.h>
#include <stdio.h>
int rename(
    const char *old,    /* existing file name */
    const char *new     /* new file name */
)
```

Description

This function changes the name of a file from `old` to `new`. It applies only to pHILE+-managed files. If a file `new` already exists, it is deleted. Certain error conditions specific to the volume type can result, as follows:

- If a file called `new` exists and is open, the call fails (except on NFS volumes).
- If `old` is open, the call fails (DOS volumes only).
- If `old` is a directory file, the call fails (DOS volumes only).

Arguments

<code>old</code>	Points to the string that contains the old filename.
<code>new</code>	Points to the string that contains the new filename.

Return Value

This function returns zero if the file is successfully renamed. If an error occurs, `errno` is set, and a nonzero value is returned.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

rewind

Resets the file position indicator.

```
#include <stdarg.h>
#include <stdio.h>
void rewind(
    FILE *stream    /* stream pointer */
)
```

Description

The `rewind()` function sets the file position indicator for the stream pointed to by `stream` to the beginning of the file. It is equivalent to:

```
(void) fseek(stream, 0L, SEEK_SET)
```

except that the stream's error indicator is cleared.

Arguments

`stream` Points to an open pREPC+ stream.
If `stream` refers to an I/O device, this function does nothing and returns a 0.

Return Value

This function does not return a value.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

`ftell`, `fgetpos`, `fsetpos`, `fseek`

scanf Reads formatted input from `stdin`.

```
#include <stdarg.h>
#include <stdio.h>
int scanf(
    const char *format,    /* format control */
    ...                   /* arguments 1 through n */
)
```

Description

The `scanf()` function is equivalent to `fscanf()` except that `scanf()` takes the input from the standard input device.

Arguments

<code>format</code>	Points to the format control string. For more information, see <code>fscanf</code> on page 3-56.
<code>...</code>	Arguments 1 through <code>n</code> point to variables where input is stored.

Return Value

This function returns `EOF` and sets `errno` if an input failure occurs before any conversion. Otherwise, it returns the number of input items assigned, which can be fewer than provided, even zero, in the event of an early matching failure.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

`printf`, `scanf`, `sscanf`

setbuf

Changes a stream's buffer.

```
#include <stdarg.h>
#include <stdio.h>
void setbuf(
    FILE *stream,    /* stream pointer */
    char *buf        /* I/O buffer */
)
```

Description

The `setbuf()` function causes a new buffer to be used for a specified stream (instead of the buffer that is currently assigned).

The buffer is assumed to have a size equal to `lc_bufsize`, which is defined in the pREPC+ Configuration Table.

The `setbuf()` function must be called after the specified stream has been opened but prior to performing any read or write operation on the stream. If `setbuf()` is called after a read or write operation, it has no effect.

You must use the `setvbuf()` call if you want additional control over buffering options.

Arguments

<code>stream</code>	Points to an open pREPC+ stream.
<code>buf</code>	Points to the new buffer. If <code>buf</code> is a null pointer, all input and output is unbuffered. This effectively eliminates buffering. Otherwise, the stream is set to fully buffered mode.

Return Value

This function does not return a value.

Notes

Callable From

- Task

Error Codes

Refer to Appendix B.

See Also

setvbuf

setlocale

Obtains or changes the program's locale.

```
#include <locale.h>
char *setlocale (
    int category          /* localization category */
    const char *locale    /* locale */
)
```

Description

The `setlocale()` function allows you to query or set certain parameters that are sensitive to the geo-political location where a program is used. For example, in Europe, the comma is sometimes used in place of the decimal point.

To query the locale or a portion thereof, you set `locale` to point to `NULL`. To change the locale, or a portion thereof, you set `locale` to point to a string that specifies the desired value.

Arguments

<code>category</code>	Specifies the localization category to be queried or changed, and must be one of the following:
<code>LC_ALL</code>	All categories.
<code>LC_COLLATE</code>	Affects the behavior of <code>strcoll()</code> and <code>strxfrm()</code> .
<code>LC_CTYPE</code>	Affects the behavior of the character-handling functions (<code>isalnum()</code> , etc.) and the multibyte functions.
<code>LC_MONETARY</code>	Affects the monetary formatting information returned by <code>localeconv()</code> .
<code>LC_NUMERIC</code>	Affects the decimal-point character for the formatted input/output functions and the string conversion functions, as well as the non-monetary formatting information returned by <code>localeconv()</code> .
<code>LC_TIME</code>	Affects the behavior of <code>strftime()</code> .

locale	Points to a string specifying the locale in which the program is used. Only one value is supported by pREPC+:
“C”	Specifies the minimal environment for C translation.

Return Value

If a pointer to a string is given for `locale` and the selection can be honored, `setlocale()` returns a pointer to the string associated with the specified category for the new locale. If the selection cannot be honored, `setlocale()` returns a null pointer and the program’s locale is not changed.

A null pointer for `locale` causes `setlocale()` to return a pointer to the string associated with the category for the program’s current locale; the program’s locale is not changed.

The pointer to string returned by `setlocale()` is such that a subsequent call with that string value and its associated category will restore that part of the program’s locale. The string pointed to will not be modified by the program, but may be overwritten by a subsequent call to `setlocale()`.

Error Codes

None.

Notes

Callable From

- Task
- ISR

See Also

`localeconv`, `strcoll`, `strxfrm`, `strftime`

setvbuf Changes a stream's buffering characteristics.

```
#include <stdarg.h>
#include <stdio.h>
int setvbuf(
    FILE *stream,    /* stream pointer */
    char *buf,       /* I/O buffer */
    int mode,        /* access mode */
    size_t size      /* buffer size */
)
```

Description

The `setvbuf()` function can be used to change either the buffer associated with a stream, the size of a stream's buffer, or the method employed for buffering the stream's data.

`setvbuf()` must be called after the specified stream has been opened, but prior to reading or writing the file. If `setbuf()` is called after a read or write operation, it has no effect.

pREPC+ allocates a buffer automatically if the `buf` parameter is `NULL` and a non-zero size is specified.

Arguments

<code>stream</code>	Points to an open pREPC+ stream.
<code>buf</code>	Points to the new buffer to be associated with the stream.
<code>mode</code>	Defines the method employed for buffering data. The possible modes are as follows:
<code>_IO_FBF</code>	Input/output is fully buffered.
<code>_IO_LBF</code>	Input/output is line buffered.
<code>_IO_NBF</code>	Input/output is not buffered.

If a stream is fully buffered, it is flushed only when it is full. If it is line buffered, the buffer is flushed either when it is full or when a new-line character is detected. `_IO_NBF` eliminates buffering; it is functionally equivalent to defining `size` equal to 0 and `buf` equal to `NULL`.

`size` Specifies the size of the buffer. `size` overrides the default buffer size defined in the pREPC+ Configuration Table.

Return Value

This function returns a 0 if it succeeds and a negative number if it fails.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

`setbuf`

sprintf Writes formatted output to a buffer.

```
#include <stdarg.h>
#include <stdio.h>
int sprintf(
    char *s,           /* buffer */
    const char *format, /* format control */
    ...                /* arguments 1 through n */
)
```

Description

The `sprintf()` function is equivalent to `fprintf()` except that `sprintf()` directs the output to a buffer pointed to by `s`.

Arguments

<code>s</code>	Points to the buffer where output is directed.
<code>format</code>	Points to the format control string. For more information, see <code>fprintf</code> on page 3-43.
<code>...</code>	Arguments 1 through <code>n</code> are written by <code>sprintf()</code> according to the specifications of the format control string.

Return Value

This function returns the number of characters written in the buffer, not counting the terminating null character.

Error Codes

None.

Notes

Callable From

- Task

See Also

`fprintf`, `printf`, `vsprintf`

srand Sets the seed for the random number generator (`rand`).

```
#include <stdlib.h>
void srand(
    unsigned int seed    /* seed value */
)
```

Description

This function works in concert with `rand()`. The `srand()` function uses `seed` as a seed for a new sequence of pseudo-random numbers to be returned by subsequent calls to `rand()`.

For a given seed value, the sequence of random numbers generated is the same. By default, a sequence of random numbers is generated using a seed value of 1.

Arguments

`seed` Specifies the seed value.

Return Value

This function does not return a value.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

`rand`

sscanf Reads formatted input from a string.

```
#include <stdarg.h>
#include <stdio.h>
int sscanf(
    const char *s,          /* string */
    const char *format,     /* format control */
    ...                     /* arguments 1 through n */
)
```

3

Description

The `sscanf()` function is equivalent to `fscanf()`, except that `sscanf()` takes the input from the string pointed to by `s`.

Arguments

<code>s</code>	Points to the string to be read.
<code>format</code>	Points to the format control string. For more information, see <code>fscanf</code> on page 3-56.
<code>...</code>	Arguments 1 through <code>n</code> point to variables where input is stored.

Return Value

This function returns the number of variable assignments. If an error occurs, the function returns EOF and sets `errno`.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

`scanf`, `fscanf`

strcat

Appends one string to another string.

```
#include <string.h>
char *strcat(
    char *s1,          /* destination string */
    const char *s2      /* source string */
)
```

3

Description

This function appends a copy of one string (*s2*) to the end of another string (*s1*). The first character in the source string overwrites the terminating null character in the destination string. If copying takes place between strings that overlap, the behavior is undefined.

Arguments

<i>s1</i>	Points to the destination string.
<i>s2</i>	Points to the source string.

Return Value

This function returns the value of *s1*.

Notes

Callable From

- Task
- ISR

Error Codes

None.

strchr

Searches a string for a character.

```
#include <string.h>
char *strchr(
    const char *s,    /* search string */
    int c             /* reference character */
)
```

Description

This function searches for the first occurrence of `c` (converted to an unsigned `char`) in the string pointed to by `s`.

Arguments

<code>s</code>	Points to the string to be searched.
<code>c</code>	Specifies the reference character.

Return Value

This function returns a pointer to the located character. If a character is not found, it returns a null pointer.

Error Codes

None.

Notes

Callable From

- Task
- ISR

strcmp

Compares two character strings.

```
#include <string.h>
int strcmp(
    const char *s1,    /* candidate string */
    const char *s2     /* candidate string */
)
```

3

Description

This function compares two character strings and returns a value that reflects whether the first string (*s1*) is greater than, equal to, or less than the second string (*s2*).

Arguments

<i>s1</i>	Points to the first string.
<i>s2</i>	Points to the second string.

Return Value

This function returns a value greater than, equal to, or less than 0, depending on whether the string pointed to by *s1* is greater than, equal to, or less than the string pointed to by *s2*.

Error Codes

None.

Notes

Callable From

- Task
- ISR

strcoll

Compares two character strings.

```
#include <string.h>
int strcoll (
    const char *s1,    /* candidate string */
    const char *s2     /* candidate string */
)
```

Description

The `strcoll()` function compares the string pointed to by `s1` to the string pointed to by `s2`. The comparison is performed relative to the current locale. (You can specify the locale by using the `setlocale()` function. See `setlocale()` on page 3-134.)

Arguments

<code>s1</code>	Points to the first string.
<code>s2</code>	Points to the second string.

Return Value

This function returns a value greater than, equal to, or less than 0, depending on whether the string pointed to by `s1` is greater than, equal to, or less than the string pointed to by `s2`.

Error Codes

None.

Notes

Callable From

- Task
- ISR

See Also

`setlocale`, `strxfrm`

strcpy

Copies one string to another string.

```
#include <string.h>
char *strcpy(
    char *s1,          /* destination string */
    const char *s2     /* source string */
)
```

Description

This function copies one string (*s2*) including the null character, into another string (*s1*). *strcpy()* continues to copy characters until it detects a null terminator. If the strings overlap, the result is unpredictable.

Arguments

<i>s1</i>	Points to the destination string.
<i>s2</i>	Points to the source string.

Return Value

This function returns the value of *s1*.

Error Codes

None.

Notes

Callable From

- Task
- ISR

strcspn

Calculates the length of a substring.

```
#include <string.h>
size_t strcspn(
    const char *s1,    /* candidate string */
    const char *s2     /* reference string */
)
```

3

Description

This function calculates the length of the maximum initial segment of a string (*s1*) which consists entirely of characters *not* in another string (*s2*).

Arguments

<i>s1</i>	Points to the string to be examined.
<i>s2</i>	Points to the reference string.

Return Value

This function returns the length of the segment.

Error Codes

None.

Notes

Callable From

- Task
- ISR

strerror Maps an error number to an error message string.

```
#include <string.h>
char *strerror (
    int errnum        /* error number */
)
```

Description

The `strerror()` function maps the error number in `errnum` to an error message string.

Arguments

`errnum` Specifies the error number.

Return Value

The `strerror()` function returns a pointer to the string. The array pointed to must not be modified by the program, but may be overwritten by a subsequent call to `strerror()`.

Error Codes

None.

Notes

Callable From

- Task
- ISR

See Also

`perror`, `errno`

strftime Places formatted time and date information into a string.

```
#include <time.h>
size_t strftime (
    char *s,                /* string */
    size_t maxsize,         /* string length */
    const char *format,     /* format control string */
    const struct tm *timeptr /* broken-down time */
)
```

3

Description

The `strftime()` function places time and date information into the string pointed to by `s` as controlled by the string pointed to by `format`.

`format` contains conversion specifiers and ordinary multibyte characters. All ordinary multibyte characters, including the terminating null character, are copied unchanged into the array. If copying takes place between objects that overlap, the behavior is undefined. No more than `maxsize` characters are placed in the array.

Arguments

<code>s</code>	Points to the string where <code>strftime()</code> places the formatted information.
<code>maxsize</code>	Specifies the length of <code>s</code> .
<code>format</code>	Contains zero or more conversion specifiers and ordinary multibyte characters. A conversion specifier consists of a <code>%</code> character followed by a character that determines the behavior of the conversion specifier. Each conversion specifier is replaced by appropriate characters as described in the following list. The appropriate characters are determined by the <code>LC_TIME</code> category of the current locale (see <code>setlocale()</code> on page 3-134) and by the values contained in the structure pointed to by <code>timeptr</code> .
<code>%a</code>	Replaced by the locale's abbreviated weekday name.
<code>%A</code>	Replaced by the locale's full weekday name.
<code>%b</code>	Replaced by the locale's abbreviated month name.
<code>%B</code>	Replaced by the locale's full month name.

<code>%c</code>	Replaced by the locale's appropriate date and time representation.
<code>%d</code>	Replaced by the day of the month as a decimal number (01-31).
<code>%H</code>	Replaced by the hour (24-hour clock) as a decimal number (00-23).
<code>%I</code>	Replaced by the hour (12-hour clock) as a decimal number (01-12).
<code>%j</code>	Replaced by the day of the year as a decimal (1-366).
<code>%m</code>	Replaced by the month as a decimal number (01-12).
<code>%M</code>	Replaced by the minute as a decimal number (00-59).
<code>%p</code>	Replaced by the locale's equivalent of the AM/PM designations associated with a 12-hour clock.
<code>%S</code>	Replaced by the second as a decimal number (00-61).
<code>%U</code>	Replaced by the week of the year, where Sunday is the first day of a week (0-52).
<code>%w</code>	Replaced by the weekday as a decimal number (0-6), where Sunday is 0.
<code>%W</code>	Replaced by the week of the year, where Monday is the first day (0-53).
<code>%x</code>	Replaced by the locale's appropriate date representation.
<code>%X</code>	Replaced by the locale's appropriate time representation.
<code>%y</code>	Replaced by the year without century as a decimal number (00-99).
<code>%Y</code>	Replaced by the year with century as a decimal number.
<code>%Z</code>	Replaced by the time zone name.
<code>%%</code>	Replaced by the percent sign.

If a conversion specifier is not one of the above, the behavior is undefined.

`timeptr` Points to the `tm` structure that contains the broken-down time. The `tm` structure is defined in the `mktime()` description on page 3-111.

Return Value

If the total number of resulting characters including the terminating null character is not more than `maxsize`, `strftime()` returns the number of characters placed into the array pointed to by `s`, not including the terminating null character. Otherwise, zero is returned and the contents of the array are indeterminate.

Error Codes

Refer to Appendix B.

3

Notes

Callable From

- Task

See Also

`setlocale`, `time`, `mktime`, `localtime`, `localtime_r`

strlen

Computes string length.

```
#include <string.h>
size_t strlen(
    const char *s    /* string */
)
```

Description

The `strlen()` function determines the length of a string (`s`), not including the terminating null character.

Arguments

`s` Points to the string to be measured.

Return Value

This function returns the computed length.

Error Codes

None.

Notes

Callable From

- Task
- ISR

strncat Appends characters to a string.

```
#include <string.h>
char *strncat(
    char *s1,          /* destination string */
    const char *s2,     /* source string */
    size_t n           /* source length */
)
```

3

Description

This function appends up to *n* characters from one string (*s2*) to the end of another string (*s1*). The first character in the source string overwrites the terminating null character in the destination string. A null character and characters that follow it in the source string are not appended.

The resulting string is always null terminated. If the length of the source string is greater than the number of characters specified in the call, only the specified number of characters (not including the null termination character) is appended.

If copying takes place between objects that overlap, the behavior is undefined.

Arguments

<i>s1</i>	Points to the destination string.
<i>s2</i>	Points to the source string.
<i>n</i>	Specifies the number of characters to append.

Return Value

This function returns the value of *s1*.

Error Codes

None.

Notes

Callable From

- Task
- ISR

strncmp

Compares characters in two strings.

```
#include <string.h>
int strncmp(
    const char *s1,    /* first string */
    const char *s2,    /* second string */
    size_t n          /* comparison size */
)
```

3

Description

This function compares up to `n` characters in two strings and returns a value that reflects whether the characters from the first string (`s1`) are greater than, equal to, or less than the characters from the second string (`s2`). Characters that follow a null character in the first string are not compared.

Arguments

<code>s1</code>	Points to the first string.
<code>s2</code>	Points to the second string.
<code>n</code>	Specifies the number of characters to compare.

Return Value

This function returns a value greater than, equal to, or less than 0, and the value depends on whether the string pointed to by `s1` is greater than, equal to, or less than the string pointed to by `s2`.

Error Codes

None.

Notes

Callable From

- Task
- ISR

strncpy

Copies characters from one string to another.

```
#include <string.h>
char *strncpy(
    char *s1,          /* destination string */
    const char *s2,     /* source string */
    size_t n           /* source length */
)
```

3

Description

This function copies up to `n` characters from one string (`s2`) to another string (`s1`). If the length of the source string is less than the specified number of characters, null characters are copied into the destination string until the specified number have been written. If the length of the source string is greater than or equal to the specified number of characters, no null characters are appended to `s1`.

Arguments

<code>s1</code>	Points to the destination string.
<code>s2</code>	Points to the source string.
<code>n</code>	Specifies the number of characters write.

Return Value

This function returns the value of `s1`.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task
- ISR

strpbrk

Searches a string for a character in a second string.

```
#include <string.h>
char *strpbrk(
    const char *s1, /* search string */
    const char *s2  /* reference string */
)
```

Description

This function locates the first occurrence in one string (*s1*) of any character in another string (*s2*).

Arguments

<i>s1</i>	Points to the string to be searched.
<i>s2</i>	Points to the reference string.

Return Value

The function returns a pointer to the first matching character. If no character from *s2* is found in *s1*, the function returns a null pointer.

Error Codes

None.

Notes

Callable From

- Task
- ISR

strchr	Searches a string for a character.
---------------	------------------------------------

```
#include <string.h>
char *strrchr(
    const char *s,    /* search string */
    int c             /* reference character */
)
```

Description

This function locates the last occurrence of `c` (converted to a `char`) in a string (`s`). The terminating null character is considered part of the string.

Arguments

s	Points to the string to be searched.
c	Specifies the reference character.

Return Value

This function returns a pointer to the character. If `c` is not found, the function returns a null pointer.

Error Codes

None.

Notes

Callable From

- Task
- ISR

strspn

Calculates specified string length.

```
#include <string.h>
size_t strspn(
    const char *s1, /* candidate string */
    const char *s2  /* reference string */
)
```

Description

The `strspn()` function computes the length of the maximum initial segment of a string (`s1`) that consists entirely of characters from another string (`s2`).

Arguments

<code>s1</code>	Points to the string to be examined.
<code>s2</code>	Points to the reference string.

Return Value

This function returns the length of the segment.

Error Codes

None.

Notes

Callable From

- Task
- ISR

strstr

Searches a string for specified characters in another string.

```
#include <string.h>
char *strstr(
    const char *s1, /* search string */
    const char *s2  /* reference string */
)
```

3

Description

The `strstr()` function locates the first occurrence in a string (`s1`) of the sequence of characters (excluding the null terminator) in another string (`s2`).

Arguments

<code>s1</code>	Points to the string to be searched.
<code>s2</code>	Points to the reference string.

Return Value

The function returns a pointer to the located string or a null pointer if the string is not found. If `s2` points to a string with zero length, the function returns `s1`.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task
- ISR

strtod

Converts a string to a double.

```
#include <stdlib.h>
double strtod(
    const char *nptr,    /* input string */
    char **endptr        /* string conversion terminator */
)
```

Description

This function converts the initial portion of a string (*nptr*) to a double representation. Leading white spaces are ignored. The string can be in scientific exponential form (for example, +123.45e+67, -123.45E+67). The `strtod()` function stops parsing the string when it detects a character that is inconsistent with a double data type.

This function sets `errno` if the converted value is out of the supported range for doubles.

Arguments

<i>nptr</i>	Points to the string to be converted.
<i>endptr</i>	An output parameter. If <i>nptr</i> is null, <code>strtod()</code> functions the same as <code>atof()</code> . If <i>nptr</i> is not null, it points to a pointer to the character in <i>str</i> that terminated the scan.

Return Value

This function returns either the converted value or 0 if no conversion occurs. No conversion occurs if the first nonwhite space in *str* is neither a digit nor a decimal point. If the correct value is outside the range of representable values, a plus or minus `HUGE_VAL` is returned. If the correct value would cause underflow, 0 is returned. If either of these two errors occurs, `errno` is set.

Notes

The pREPC+ library returns double values (including floating point) in the CPU register pair designated by the compiler to receive a return value of type `double` from a function call when a hardware floating point is *not* selected. Additionally, if

the FPU bit is set in the processor type entry of the Node Configuration Table, the pREPC+ library also places the floating point value in the floating point register designated by the compiler to receive a return value of type `double` when a hardware floating point *is* selected.

Callable From

- Task

Error Codes

Refer to Appendix B.

See Also

`atoi`, `atol`, `atof`

strtok

Searches a string for tokens.

```
#include <string.h>
char *strtok(
    char *s1,          /* search string */
    const char *s2      /* delimiter(s) */
)
```

Description

A series of calls to the `strtok()` function breaks a string (`s1`) into a sequence of *tokens*, each of which is delimited by a character in a second string (`s2`). A token is a sequence of one or more characters. The first call to `strtok()` has `s1` as its first argument, and is followed by calls with a null pointer as their first argument.

On the first call to `strtok()`, `s1` is passed as a pointer to the string to be searched, and `s2` is passed as a pointer to the string that contains the delimiters. The first call searches for the first character in `s1` that is not found in the delimiter set. If such a character is found, it is the start of the first token. If the first call fails to find a character that is not a delimiter, there are no tokens, and a null pointer is returned.

The function then continues the search of `s1` for a character that *is* contained in the delimiter set. If no such character is located, the current token extends to the end of `s1`, and subsequent calls to the function return a null pointer. If a delimiter is located, a null character that terminates the current token overwrites the delimiter. The function saves the pointer to the character that follows. This is where the next search for a token starts.

In subsequent calls, the first argument should be a null pointer. The search for the next token begins from the saved pointer and behaves as described in the preceding paragraph.

The search string `s2` can be changed between calls. This allows `strtok()` to continue to parse the string with a different set of delimiters.

Arguments

- | | |
|-----------------|---|
| <code>s1</code> | Points to the string to be searched. |
| <code>s2</code> | Points to the string containing token delimiters. |

Return Value

The function returns a pointer to the first character of a token. If no token exists, the function returns a null pointer.

Error Codes

None.

Notes

3

Callable From

- Task

strtol

Converts a string to a long integer.

```
#include <stdlib.h>
long strtol(
    const char *nptr,    /* string */
    char **endptr,       /* string conversion terminator */
    int base             /* conversion base */
)
```

Description

The `strtol()` function converts the initial portion of a string (`nptr`) to long int representation, according to the radix specified by `base`. This function ignores leading white spaces, and the string can contain either a `+` or a `-`.

Arguments

<code>nptr</code>	Points to the string to be converted.
<code>endptr</code>	An output parameter. If <code>endptr</code> is null, this function is equivalent to the <code>atol()</code> function. If <code>endptr</code> is not null, it points to a pointer to the character in <code>str</code> that terminated the scan.
<code>base</code>	Specifies the base of the number system assumed by the function. <code>base</code> must be either 0 or within the range 2 through 36. If it is 0, the string itself is used to determine its base. If <code>nptr</code> begins with the character 0, base eight is assumed; if <code>nptr</code> begins with 0x or 0X, base sixteen is assumed; otherwise base ten is assumed.

Return Value

This function returns the converted value. If the conversion fails, this function returns a 0 and sets `errno`.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

`atoi`, `atof`

strtoul

Converts a string to an unsigned long.

```
#include <stdlib.h>
unsigned long strtoul(
    const char *nptr,    /* string */
    char **endptr,       /* string conversion terminator*/
    int base              /* conversion base */
)
```

Description

This function is equivalent to `strtol()` except that the minus sign (-) is not a valid character to `strtoul()` and the string is converted to an unsigned long representation.

Arguments

<code>nptr</code>	Points to the string to be converted.
<code>endptr</code>	An output parameter. If <code>endptr</code> is null, this function is equivalent to the <code>atol()</code> function. If <code>endptr</code> is not null, it points to a pointer to the character in <code>nptr</code> that terminated the scan.
<code>base</code>	Specifies the base of the number system assumed by the function. <code>base</code> must be either 0 or within the range 2 through 36. If it is 0, the string itself is used to determine its base. If <code>nptr</code> begins with the character 0, base eight is assumed; if <code>nptr</code> begins with 0x or 0X, base sixteen is assumed; otherwise base ten is assumed.

Return Value

This function returns the converted value. If an error occurs, this function returns a 0, and sets `errno`.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

atol

strxfrm Transforms a string so that it can be used by the `strcmp()` function.

```
#include <string.h>
size_t strxfrm (
    char *s1,    /* destination string */
    const char *s2 /* source string */
    size_t n     /* destination string length */
)
```

Description

The `strxfrm()` function transforms the string pointed to by `s2` and places the resulting string into the array pointed to by `s1`. The transformation is such that if the `strcmp()` function is applied to two transformed strings, it returns a value greater than, equal to, or less than zero, corresponding to the result of the `strcoll()` function applied to the same two original strings. No more than `n` characters are placed into the resulting array pointed to by `s1`, including the terminating null character. If `n` is zero, `s1` is permitted to be a null pointer. If copying takes place between objects that overlap, the behavior is undefined.

The main use for this function is in foreign language environments that do not use the ASCII collating sequence.

Arguments

<code>s1</code>	Points to the array where <code>strxfrm()</code> places the resulting string.
<code>s2</code>	Points to the string to be transformed.
<code>n</code>	Specifies the number of characters to be placed in <code>s1</code> .

Return Value

The `strxfrm()` function returns the length of the transformed string (not including the terminating null character.) If the value is `n` or more, the contents of the array pointed to by `s1` are indeterminate.

Error Codes

None.

Notes

Callable From

- Task
- ISR

See Also

`setlocale`, `strcoll`, `strcmp`

time Obtains the current calendar time.

```
#include <time.h>
time_t time (
    time_t *timer    /* buffer */
)
```

Description

The `time()` function determines the current calendar time, expressed as the number of seconds since midnight January 1, 1970.

Arguments

`timer` Points to the buffer where `time()` can store the current calendar time.

Return Value

The `time()` function returns the implementation's best approximation of the current calendar time. The value `(time_t) -1` is returned if the calendar time is not available. If `timer` is not a null pointer, the return value is also assigned to the object it points to.

Error Codes

Refer to Appendix B.

Notes

This function invokes the pSOS+ service call `tm_get()` to obtain the current time.

Callable From

- Task
- ISR

See Also

`tm_get`, `mktime`, `localtime`, `localtime_r`

tmpfile

Creates a temporary file.

```
#include <stdarg.h>
#include <stdio.h>
FILE *tmpfile(
    void
)
```

Description

The `tmpfile()` function creates a temporary file that is automatically removed when it is closed or when the creating task calls `exit()`. Temporary files are opened in mode `wb+`.

The name of the temporary file created is obtained by generating an internal call to the pREPC+ function `tmpname()`.

Return Value

This function returns a pointer to the stream of the file or a null pointer if no file is created. If an error occurs, the function sets `errno`.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

`tmpname`

tmpnam Generates a temporary filename.

```
#include <stdarg.h>
#include <stdio.h>
char *tmpnam(
    char *s    /* string root */
)
```

3

Description

The `tmpnam()` function generates a string that is intended to be used as a filename. `tmpnam()` generates up to 1000 unique names for each task. The pREPC+ library does not maintain a list of `tmpnames` in use. After 1000 names have been generated, the sequence starts over.

A file with a name generated by `tmpnam()` is not necessarily a temporary file. To be treated as a temporary file, it must be created by `tmpfile()`.

Arguments

s Points to the string where `tmpnam()` stores the filename.

When `tmpnam()` is called, `s` should consist of the initial part of a valid pathname. `tmpnam()` adds a slash (/), followed by a `T` (or `G` if the creating task is global). The `T` (or `G`) is followed by the lower 16-bits of the caller's task ID, which is followed by a decimal point and a decimal number within the range 0 through 999.

For example, assume that `s` points to the string `0.0`; the caller's tid is `00000002`; and the caller has previously called `tmpnam()` 12 times. The generated name is `0.0/T0002.012`.

Return Value

This function returns a pointer to the generated name.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

tolower

Converts a character to lowercase.

```
#include <ctype.h>
int tolower(
    int c    /* character */
)
```

Description

This function converts an uppercase letter to lowercase.

Arguments

c Specifies the character to be converted.

Return Value

This function returns the converted character. If **c** is not an uppercase character, the function returns the character unchanged.

Error Codes

None.

Notes

Callable From

- Task
- ISR

toupper Converts a character to uppercase.

```
#include <ctype.h>
int toupper(
    int c    /* character */
)
```

Description

This function converts a lowercase letter to uppercase.

Arguments

c Specifies the character to be converted.

Return Value

This function returns the converted character. If **c** is not a lowercase character, it is returned unchanged.

Error Codes

None.

Notes

Callable From

- Task
- ISR

ungetc

Ungets a character.

```
#include <stdarg.h>
#include <stdio.h>
int ungetc(
    int c,          /* character */
    FILE *stream    /* stream pointer */
)
```

3

Description

The `ungetc()` function pushes a character (`c`), converted to an unsigned char, back to the specified stream. The character is returned on the next read operation on the stream. A call to `fseek()`, `fsetpos()`, `rewind()`, or `fflush()` ignores the character.

The stream's position indicator is not changed by this call.

Arguments

<code>c</code>	Specifies the character to be pushed.
<code>stream</code>	Points to an open pREPC+ stream.

Return Value

This function returns the contents of `c`. If an error occurs, the function returns EOF and sets `errno`.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

`putc`, `getc`

vfprintf

Writes formatted output to a stream.

```

#include <stdarg.h>
#include <stdio.h>
int vfprintf(
    FILE *stream,          /* stream pointer */
    const char *format,    /* format control */
    va_list arg            /* argument list */
)

```

3

Description

The `vfprintf()` function is equivalent to `fprintf()`, with the variable argument list replaced by `arg`, which should have been initialized by the `va_start` macro (and possibly subsequent `va_arg` calls). The `vfprintf` function does not invoke the `va_end` macro.

Arguments

<code>stream</code>	Points to an open pREPC+ stream.
<code>format</code>	Points to the format control string. For more information, see <code>fprintf</code> on page 3-43.
<code>arg</code>	A list of arguments to be written according to the specifications of the format control string.

Return Value

This function returns the number of characters written. If a write error occurs, this function returns a negative number and sets `errno`.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

`fprintf`, `vprintf`, `vsprintf`

vprintf

Writes formatted output to `stdout`.

```
#include <stdarg.h>
#include <stdio.h>
int vprintf(
    const char *format,    /* format control */
    va_list arg           /* argument list */
)
```

3

Description

The `vprintf()` function is equivalent to `printf()`, with the variable argument list replaced by `arg`, which should have been initialized by the `va_start` macro (and possibly subsequent `va_arg` calls). The `vfprintf` function does not invoke the `va_end` macro.

Arguments

<code>format</code>	Points to the format control string. For more information, see <code>fprintf</code> on page 3-43.
<code>arg</code>	A list of arguments to be written according to the specifications of the format control string.

Return Value

This function returns the number of characters written. If a write error occurs, this function returns a negative number and sets `errno`.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

`printf`, `vfprintf`, `fprintf`, `vsprintf`

vsprintf

Writes formatted output to a buffer.

```
#include <stdio.h>
#include <stdarg.h>
int vsprintf(
    char *s,                /* target buffer */
    const char *format,     /* format control */
    va_list char arg        /* argument list */
)
```

3

Description

The `vsprintf()` function is equivalent to `sprintf()`, with the variable argument list replaced by `arg`, which should have been initialized by the `va_start` macro (and possibly subsequent `va_arg` calls). The `vfprintf` function does not invoke the `va_end` macro.

Arguments

<code>s</code>	Points to the buffer where output is directed.
<code>format</code>	Points to the format control string. For more information, see <code>fprintf</code> on page 3-43.
<code>arg</code>	A list of arguments to be written according to the specifications of the format control string.

Return Value

This function returns the number of characters written. If a write error occurs, this function returns a negative number and sets `errno`.

Error Codes

Refer to Appendix B.

Notes

Callable From

- Task

See Also

`printf`, `sprintf`, `fprintf`

wcstombs Converts a wide character string into a multibyte character string.

```
#include <stdlib.h>
size_t wcstombs (
    char *s,                /* result string */
    const wchar_t *pwcs,    /* wide string */
    size_t n                /* size of result string */
)
```

3

Description

The `wcstombs()` function converts a sequence of codes that correspond to multibyte characters from the array pointed to by `pwcs` into a sequence of multibyte characters that begins in the initial shift state and stores these multibyte characters in the array pointed to by `s`, stopping if a multibyte character would exceed the limit of `n` total bytes or if a null character is stored. Each call is converted as if by a call to the `wctomb()` function, except that the shift state of the `wctomb()` function is not affected.

No more than `n` bytes will be modified in the array pointed to by `s`. If copying takes place between objects that overlap, the behavior is undefined.

Arguments

<code>s</code>	Points to the array where <code>wcstombs()</code> stores the converted string.
<code>pwcs</code>	Points to the string to be converted.
<code>n</code>	Specifies the size of <code>s</code> .

Return Value

If a code is encountered that does not correspond to a valid multibyte character, this function returns `(size_t) -1.` Otherwise, it returns the number of bytes modified, not including a terminating null character, if any.

Error Codes

None.

Notes

Callable From

- Task
- ISR

See Also

wctomb, mbtowc, mbstowcs

wctomb Converts a wide character into its multibyte character equivalent.

```
#include <stdlib.h>
int wctomb (
    char *s,          /* result character */
    wchar_t wchar     /* wide character */
)
```

3

Description

The `wctomb()` function determines the number of bytes needed to represent the multibyte character corresponding to the code whose value is `wchar` (including any change in shift state). It stores the multibyte character representation in the array object pointed to by `s` (if `s` is not a null pointer). At most `MB_CUR_MAX` characters are stored. If the value of `wchar` is zero, the `wctomb()` function is left in its initial state.

Arguments

<code>s</code>	Points to the array where <code>wctomb()</code> stores the converted character.
<code>wchar</code>	Points to the character to be converted.

Return Value

If `s` is a null pointer, this function returns a nonzero or zero value, if multibyte character encodings, respectively, do or do not have state-dependent encodings. If `s` is not a null pointer, this function returns -1 if the value of `wchar` does not correspond to a valid multibyte character, or returns the number of bytes that are contained in the multibyte character corresponding to the value of `wchar`.

In no case will the value returned be greater than the value of the `MB_CUR_MAX` macro.

Error Codes

None.

Notes

Callable From

- Task
- ISR

See Also

`mbtowc`, `wctombs`, `mbstowcs`

4

pNA+ System Calls

This chapter provides detailed information on each system call in the pNA+ component of pSOSystem. The calls are listed alphabetically, with a multipage section of information for each call. Each call's section includes its syntax, a detailed description, its arguments, its return value, and any error codes that it can return. Where applicable, the section also includes the headings "Notes," "Usage," and "See Also." "Notes" contains any important information not specifically related to the call description; "Usage" provides detailed usage information; and "See Also" indicates other calls that have related information.

Structures described in this chapter are also defined in the file `<pna.h>`. Structures must be word-aligned and must not be packed.

If you need to look up a system call by its functionality, refer to Appendix A, "Tables of System Calls," which lists the calls alphabetically by component and provides a brief description of each call.

For more information on error codes, refer to Appendix B, "Error Codes," which lists the codes numerically and shows which pSOSystem calls are associated with each one.

accept

Accepts a connection on a socket.

```
#include <pna.h>
long accept(
    int s,                      /* socket descriptor */
    struct sockaddr_in *addr,    /* socket structure */
    int *addrlen                 /* socket structure size */
)
```

Description

This call is used to accept a connection request that the specified socket receives from a foreign socket. Servers use `accept()` with connection-oriented or stream (TCP) sockets.

Before `accept()` is called, the socket must be set up to receive a connection request by issuing the `listen()` system call. `accept()` extracts the first connection request on the queue of pending connections; creates a new socket with the same properties as the original socket; completes the connection between the foreign socket and the new socket; and returns a descriptor for the new socket. The new returned socket descriptor is used to read from and write data to the foreign socket. It is not used to accept more connections. The original socket remains open for accepting further connections.

If no pending connections exist on the queue and the socket is not marked as non-blocking, `accept()` blocks the caller until a connection is present. If the socket is marked non-blocking and no pending connections are present on the queue, `accept()` returns an error.

Upon return, `accept()` stores the address of the connected socket in the specified socket address structure.

Arguments

<code>s</code>	Specifies the socket on which to accept a connection request.
----------------	---

`addr` Points to a structure of type `sockaddr_in` where `accept()` stores the address of the connected socket. The structure `sockaddr_in` is defined in the file `<pna.h>` and has the following format:

```
struct sockaddr_in {
    short sin_family;          /* must be AF_INET */
    unsigned short sin_port;   /* 16-bit port number */
    struct in_addr sin_addr;   /* 32-bit IP address */
    char sin_zero[8];         /* must be 0 */
};
```

This structure cannot be packed.

`addrlen` Points to an integer equal to 16, which is the size in bytes of the `sockaddr_in` structure.

Return Value

If this call succeeds, it returns a non-negative integer that is a descriptor for the accepted socket. It returns -1 on error.

Error Codes

Hex	Mnemonic	Description
0x5009	EBADS	The socket descriptor is invalid.
0x5017	ENFILE	An internal table has run out of space.
0x5016	EINVALID	An argument is invalid.
0x5023	EWOULDBLOCK	This operation would block and the socket is marked non-blocking.
0x5034	ECONNABORTED	The connection has been aborted by the peer.
0x5037	ENOBUFS	An internal buffer is required but cannot be allocated.
0x502D	EOPNOTSUPP	The requested operation is not valid for this type of socket.

See Also

`bind`, `connect`, `listen`, `select`, `socket`

add_ni Adds a network interface.

```
#include <pna.h>
long add_ni(
    struct ni_init *ni    /* network interface */
)
```

Description

This system call is used to dynamically add a network interface to the pNA+ network manager. The characteristics of the network interface are specified in the data structure pointed to by `ni`.

This routine calls the network interface driver's `NI_INIT` routine for driver initialization.

Arguments

`ni` Points to an `ni_init` structure. The structure `ni_init` is defined in the file `<pna.h>` and has the following format:

```
struct ni_init {
    int (*entry)();    /* address of NI entry point */
    int ipadd;         /* IP address */
    int mtu;           /* maximum transmission length */
    int hwalen;        /* length of hardware address */
    int flags;         /* interface flags */
    int subnetaddr;    /* subnet mask */
    int dstipaddr;     /* Dest. address in Point-to Point NI */
    int reserved[1];   /* reserved for future use */
};
```

This structure cannot be packed. The `flags` element can contain one or more of the following symbolic constants (defined in `pna.h`), using the syntax:

IFF_BROADCAST IFF_RAWMEM	
Symbolic Constant	Description
<code>IFF_BROADCAST</code>	NI can broadcast.
<code>IFF_EXTLOOPBACK</code>	NI uses external loopback.
<code>IFF_INITDOWN</code>	NI must be initialized in DOWN state. Default is UP state.

IFF_MULTICAST	NI supports multicast.
IFF_NOARP	NI does not have ARP.
IFF_POINTTOPOINT	NI is point-to-point driver.
IFF_POLL	pNA+ polls NI at regular intervals
IFF_RAWMEM	NI passes packets as mblks.
IFF_UNNUMBERED	NI is an unnumbered link.

Return Value

This system call returns the pNA+ interface number of the new network interface if successful; otherwise it returns -1.



Error Codes

Hex	Mnemonic	Description
0x5016	EINVALID	An argument is invalid.
0x5017	ENFILE	An internal table has run out of space.
0x5046	ENIDOWN	NI_INIT returned -1.
0x5047	ENMTU	The MTU is invalid.
0x5048	ENHWL	The hardware length is invalid.

See Also

Network Interfaces and Configuration Tables, *pSOSystem Programmer's Reference*

bind

Binds an address to a socket.

```
#include <pna.h>
long bind(
    int s,                      /* socket descriptor */
    struct sockaddr_in *addr,   /* socket structure */
    int addrlen                 /* structure size */
)
```

Description

This system call is used to assign (or bind) an address (a 32-bit internet address and a 16-bit port number) to a socket. A socket is created without an address and cannot be used to receive data until it is assigned one. Raw IP sockets are an exception. If they are unbound then they receive all packets regardless of the packet's addresses.

To simplify address binding, a wildcard internet address is supported to free the user from needing to know the local internet address. It also makes programs more portable. When the internet address is specified as the symbolic constant `INADDR_ANY`, pNA+ interprets it as any valid address. This allows the socket to receive data regardless of its node's internet address. For example, if a socket is bound to `<INADDR_ANY, 10>` and it resides on a node that is attached to networks 90.0.0.2 and 100.0.0.3, the socket can receive data addressed to `<90.0.0.2, 10>` or `<100.0.0.3, 10>`.

Arguments

<code>s</code>	Specifies the socket to which the address is bound.
<code>addr</code>	Points to a structure of type <code>sockaddr_in</code> that stores the socket attributes to be bound to the socket. The structure <code>sockaddr_in</code> is defined in the file <code><pna.h></code> and has the following format:

```
struct sockaddr_in {
    short sin_family;          /* must be AF_INET */
    unsigned short sin_port;   /* 16-bit port number */
    struct in_addr sin_addr;    /* 32-bit IP address */
    char sin_zero[8];          /* must be 0 */
};
```

This structure cannot be packed.

`addrlen` Specifies the size in bytes of the `sockaddr_in` structure and must be 16.

Return Value

This system call returns 0 if successful, otherwise it returns -1.

Error Codes

Hex	Mnemonic	Description
0x5009	EBADS	The socket descriptor is invalid.
0x5016	EINVALID	An argument is invalid.
0x5030	EADDRINUSE	The specified address is already in use.
0x5031	EADDRNOTAVAIL	The specified address is not available.

See Also

`connect`, `getsockname`, `listen`, `socket`

close

Closes a socket descriptor.

```
#include <pna.h>
long close(
    int s    /* socket descriptor */
)
```

Description

The `close()` call discards the specified socket descriptor. If it is the last descriptor associated with the socket, the socket is deleted and, unless the `SO_LINGER` socket option is set, any data queued at the socket is discarded. Refer to the `setsockopt()` pNA+ call for a discussion of the `SO_LINGER` option.

As a special case, if the specified socket descriptor is equal to 0, `close()` closes all socket descriptors that have been allocated to the calling task.

Arguments

`s` Specifies the socket to be closed.

Return Value

This system call returns 0 if successful, otherwise it returns -1.

Error Codes

Hex	Mnemonic	Description
0x5009	EBADS	The socket descriptor is invalid.
0x5016	EINVALID	An argument is invalid.

See Also

`socket`, `setsockopt`

connect Initiates a connection on a socket.

```
#include <pna.h>
long connect(
    int s,                      /* socket descriptor */
    struct sockaddr_in *addr,   /* socket attributes */
    int addrlen                 /* attribute size */
)
```

4

Description

This system call is used to establish an association between a local socket and a foreign socket.

Generally, a stream socket connects only once. A datagram socket can use `connect()` multiple times to change its association. A datagram socket can dissolve the association by connecting to an invalid address, such as the null address `INADDR_ANY` defined in `pna.h`.

If a stream socket is specified, `connect()` initiates a connection request to the foreign socket. The caller is blocked until a connection is established, unless the socket is non-blocking.

If a datagram socket is specified, `connect()` associates the socket with the socket address supplied. This address is used by future `send()` calls to determine the datagram's destination. This is the only address from which datagrams can be received.

If a raw socket is specified, `connect()` associates the socket with the socket address supplied. This address is used by future `send()` calls to determine the datagram's destination. This is the only address from which datagrams can be received.

Arguments

`s` Specifies the local socket.

addr Points to a `sockaddr_in` structure that contains the address of the foreign socket. The structure `sockaddr_in` is defined as follows:

```
struct sockaddr_in {
    short sin_family;          /* must be AF_INET */
    unsigned short sin_port;   /* 16-bit port number */
    struct in_addr sin_addr;   /* 32-bit IP address */
    char sin_zero[8];          /* must be 0 */
};
```

This structure cannot be packed.

addrlen Specifies the size in bytes of the `sockaddr_in` structure and must be 16.

Return Value

This system call returns 0 if successful, otherwise it returns -1.

Error Codes

Hex	Mnemonic	Description
0x5009	EBADS	The socket descriptor is invalid.
0x5016	EINVALID	An argument is invalid.
0x5024	EINPROGRESS	The socket is non-blocking and the connection cannot be completed immediately.
0x5025	EALREADY	The socket is non-blocking and a previous connection attempt has not yet been completed.
0x502F	EAFNOSUPPORT	The <code>sin_family</code> member of <code>sockaddr_in</code> isn't <code>AF_INET</code> .
0x5030	EADDRINUSE	The address specified is already in use.
0x5031	EADDRNOTAVAIL	The specified address is not available.
0x5037	ENOBUFS	An internal buffer is required but can't be allocated.
0x5038	EISCONN	The socket <code>s</code> is already connected.
0x503C	ETIMEDOUT	Connection timed out.
0x503D	ECONNREFUSED	The attempt to connect was refused.

See Also

`accept`, `close`, `connect`, `getsockname`, `select`, `socket`

get_id

Gets a task's user ID and group ID.

```
#include <pna.h>
long get_id(
    long *userid,    /* task user ID */
    long *groupid,   /* task group ID */
    long *groups;    /* must be zero */
)
```

Description

This system call obtains the user ID and group ID of the calling task. These IDs are used for accessing NFS servers. The user ID and group ID are set by using the `set_id()` call. Default values for these IDs are defined in the pNA+ Configuration Table.

Arguments

<code>userid</code>	Points to a long variable where <code>get_id()</code> stores the calling task's user ID.
<code>groupid</code>	Points to a long variable where <code>get_id()</code> stores the calling task's group ID.
<code>groups</code>	Zero must be passed as a third argument (which is currently ignored).

Return Value

This system call returns 0 if successful, otherwise it returns -1.

Error Codes

None.

See Also

`set_id`

getpeername Gets the address of a connected peer.

```
#include <pna.h>
long getpeername(
    int s,                      /* socket descriptor */
    struct sockaddr_in *addr,    /* socket attributes */
    int *addrlen                /* socket structure size */
)
```

4

Description

The `getpeername()` call obtains the address of the *peer* connected to the specified socket. The peer is the socket at the other end of the connection.

Arguments

<code>s</code>	Specifies the original socket.
<code>addr</code>	Points to a <code>sockaddr_in</code> structure in which <code>getpeername()</code> stores the address of the peer socket. The structure <code>sockaddr_in</code> is defined as follows: <pre>struct sockaddr_in { short sin_family; /* must be AF_INET */ unsigned short sin_port; /* 16-bit port number */ struct in_addr sin_addr; /* 32-bit IP address */ char sin_zero[8]; /* must be 0 */ };</pre> <p>This structure cannot be packed.</p>
<code>addrlen</code>	Points to an integer equal to 16, which is the size in bytes of the <code>sockaddr_in</code> structure.

Return Value

This system call returns 0 if successful, otherwise it returns -1.

Error Codes

Hex	Mnemonic	Description
0x5009	EBADS	The socket descriptor is invalid.
0x5037	ENOBUFS	An internal buffer is required, but cannot be allocated.
0x5039	ENOTCONN	The socket is not connected.

See Also

`accept`, `bind`, `getsockname`, `socket`

getsockname Gets the address that is bound to a socket.

```
#include <pna.h>
long getsockname(
    int s,                      /* socket descriptor */
    struct sockaddr_in *addr,    /* socket attributes */
    int *addrlen                 /* size of sockaddr_in */
)
```

4

Description

The `getsockname()` call obtains the address bound to the specified socket.

Arguments

<code>s</code>	Specifies the socket.
<code>addr</code>	Points to a <code>sockaddr_in</code> structure in which <code>getsockname()</code> stores the address bound to the socket. The <code>sockaddr_in</code> structure is defined as follows: <pre>struct sockaddr_in { short sin_family; /* must be AF_INET */ unsigned short sin_port; /* 16-bit port number */ struct in_addr sin_addr; /* 32-bit IP address */ char sin_zero[8]; /* must be 0 */ };</pre> <p>This structure cannot be packed.</p>
<code>addrlen</code>	Points to an integer equal to 16, which is the size in bytes of the <code>sockaddr_in</code> structure.

Return Value

This system call returns 0 if successful, otherwise it returns -1.

Error Codes

Hex	Mnemonic	Description
0x5009	EBADS	The socket descriptor is invalid.
0x5037	ENOBUFS	An internal buffer is required, but cannot be allocated.

See Also

`bind`, `getpeername`, `socket`

getsockopt Gets options on a socket.

```
#include <pna.h>
long getsockopt(
    int s,          /* socket descriptor */
    int level,      /* SOL_SOCKET, IPPROTO_IP, */
                  /* or IPPROTO_TCP */
    int optname,    /* retrieval option */
    char *optval,   /* return buffer */
    int *optlen     /* input/output buffer size */
)
```

4

Description

The `getsockopt()` system call obtains the status of options associated with the specified socket. Socket level, IP protocol level, or TCP protocol level options may be retrieved.

Arguments

<code>s</code>	Specifies the socket.
<code>level</code>	Specifies the level of the option to be queried and must be set to <code>SOL_SOCKET</code> for socket level operations, <code>IPPROTO_IP</code> for IP protocol level operations, or <code>IPPROTO_TCP</code> for TCP protocol level operations.
<code>optname</code>	Specifies the option to be queried, and uses a symbolic constant. The symbolic constants available for each level are provided below and in <code><pna.h></code> .
<code>optval</code>	Points to a buffer where <code>getsockopt()</code> stores the value for the requested option. For most options, an <code>int</code> is returned in the buffer pointed to by <code>optval</code> . A nonzero value means the option is set, and a 0 means the option is off.
<code>optlen</code>	An input-output parameter. On input, it should contain the size of the buffer pointed to by <code>optval</code> . On output, it contains the actual size of the value returned in the <code>optval</code> buffer.

Socket Level Options

`level` must be set to `SOL_SOCKET` for socket level operations. The `optname` value can be one of the following:

<code>SO_BROADCAST</code>	Allows broadcast datagrams on a socket.
<code>SO_DONTROUTE</code>	Indicates that the outgoing messages should not be routed. Packets directed to unconnected networks are dropped.
<code>SO_ERROR</code>	Returns the pending error and clears the error status.
<code>SO_KEEPALIVE</code>	Keeps the connection alive by periodically transmitting a packet over socket <code>s</code> .
<code>SO_LINGER</code>	Controls the action taken when unsent messages are queued on a socket and a <code>close()</code> is executed. If the socket is a stream socket and <code>SO_LINGER</code> is set (<code>l_onoff</code> set to 1), the calling task blocks until it can transmit the data or until a timeout period expires. If <code>SO_LINGER</code> is disabled (<code>l_onoff</code> set to 0), the socket is deleted immediately. <code>SO_LINGER</code> uses the <code>linger</code> structure, which is defined as follows: <pre>struct linger { int l_onoff; /* on/off option */ int l_linger; /* linger time in secs.*/ }</pre> <p>This structure cannot be packed.</p>
<code>SO_OOBINLINE</code>	Requests that out-of-band data go into the normal data input queue as received; it then is accessible with <code>recv()</code> calls without the <code>MSG_OOB</code> flag.
<code>SO_RCVBUF</code>	Adjusts the normal buffer size allocated for a socket input buffer.
<code>SO_REUSEADDR</code>	Indicates that local addresses can be reused in a <code>bind()</code> call.
<code>SO_REUSEPORT</code>	Indicates that local addresses can be reused in a <code>bind()</code> call. For more information, see section 4.4.3 of the Network Programming chapter in <i>pSOSystem System Concepts</i> .
<code>SO_SNDBUF</code>	Adjusts the normal buffer size allocated for a socket output buffer.
<code>SO_TYPE</code>	Returns the type of socket.

TCP Level Option

`level` must be set to `IPPROTO_TCP` for TCP protocol level operations. The argument `optname` can have the following value:

<code>TCP_KEEPA_LIVE_</code> <code>CNT</code>	Number of Keepalive strobos. Upon expiration of the Keepalive idle timer TCP will send a number of strobos separated by a fixed interval. If the other end fails to respond to the strobos (special TCP segments) then the TCP connection will be terminated. The default number of strobos in pNA+ is set to 8. Only valid if the <code>SO_KEEPA_LIVE</code> option is set above.
<code>TCP_KEEPA_LIVE_</code> <code>IDLE</code>	Keepalive idle time in TCP. If the connection has been idle for this time, the timer will expire causing TCP to send a special segment forcing the other end to respond. On demand-dial links for example the timer may be set long enough so as not to cause unnecessary traffic. The default in pNA+ is 120 minutes. The timer is in seconds. Only valid if the <code>SO_KEEPA_LIVE</code> option is set above.
<code>TCP_KEEPA_LIVE_</code> <code>INTVL</code>	Keepalive strobe interval. The strobos sent out by TCP upon expiration of the Keepalive idle timer are separated by a fixed time interval. The default interval between the strobos is set to 75 seconds in pNA+. The time interval is in seconds. Only valid if the <code>SO_KEEPA_LIVE</code> option is set above.
<code>TCP_MSL</code>	Maximum Segment Lifetime in TCP. This controls the <code>TIME_WAIT</code> or <code>2MSL</code> timer in TCP which is set to twice the value of the MSL. The timer is used to validate connection termination and transmits remaining data in the send queue. It is a safeguard against sequence numbers being overlapped. If set to a low value it allows the sockets to be quickly deleted. The default in pNA+ is 30 seconds. The timer is in seconds.
<code>TCP_NODELAY</code>	Disables delay acknowledgment algorithm. Data is sent immediately over the network instead of waiting for the window to be full.

IP Level Options

`level` must be set to `IPPROTO_IP` for IP protocol level operations. The argument `optname` can have one of the following values:

<code>IP_HDRINCL</code>	Specifies that the IP Header will be included in the output packets. The following fields will be set by pNA+ if they are set to 0 in the included IP header: IP identification number and IP source address. The fragmentation offset and checksum fields are always computed by pNA+. The rest of the IP header fields must be set appropriately.
<code>IP_MULTICAST_IF</code>	Specifies the outgoing interface for multicast packets. For this option, <code>optval</code> is a pointer to <code>struct in_addr</code> . If set to <code>INADDR_ANY</code> then the routing table will be used to select an appropriate interface.
<code>IP_MULTICAST_INTF</code>	Specifies the outgoing interface for multicast packets. The outgoing interface is defined by its interface number. The <code>optval</code> is a pointer to a long. If set to -1 then the routing table will be used to select an appropriate interface, which is also the default case.
<code>IP_MULTICAST_LOOP</code>	Specifies whether or not to loopback multicast packets. <code>optval</code> is a pointer to an unsigned char. By default the packets are looped back (<code>IP_DEFAULT_MULTICAST_LOOP</code> .) A value of 0 disables loopback.
<code>IP_MULTICAST_TTL</code>	Specifies the time-to-live for outgoing IP multicast datagrams. <code>optval</code> is a pointer to an unsigned char. The default is <code>IP_DEFAULT_MULTICAST_TTL</code> .

Return Value

This system call returns 0 if successful, otherwise it returns -1.

Error Codes

Hex	Mnemonic	Description
0x5009	EBADS	The socket descriptor is invalid.
0x502A	ENOPROTOOPT	The optname or level is not valid.

See Also

setsockopt, socket

ioctl Performs control operations on a socket.

```
#include <pna.h>
long ioctl(
    int s,          /* socket descriptor */
    int cmd,        /* socket operation */
    char *arg       /* operation argument */
)
```

Description

The `ioctl()` system call is used to perform control operations on the specified socket.

Arguments

<code>s</code>	Specifies the socket.
<code>cmd</code>	Specifies the operation and is a symbolic constant. All permissible <code>cmd</code> values, except MIB-related operations, are defined in <code><pna.h></code> . MIB-related <code>cmd</code> values are defined in <code><pna_mib.h></code> . Operation descriptions are provided below.
<code>arg</code>	Points to a data structure that is dependent on the value of <code>cmd</code> and contains additional information needed by <code>ioctl()</code> to perform the operation.

Operations

System-Related Operations

Operation	Description
<code>FIOASYNC</code>	Controls whether or not the user-provided signal handler is called when events related to the socket occur (for example, if the socket receives urgent data). If the integer pointed to by <code>arg</code> equals 1, signalling is enabled. If the integer pointed by <code>arg</code> equals 0, signalling is disabled.
<code>FIOGETOWN</code>	Identifies the owner of the socket. The task ID of the owner task is returned in the integer variable pointed to by <code>arg</code> .

Operation	Description
FIONBIO	Sets the blocking mode of the socket. If the integer pointed to by <code>arg</code> equals 1, the socket is set to operate in non-blocking mode. If the integer pointed to by <code>arg</code> equals 0, the socket is set to operate in blocking (default) mode. Normally, socket operations that cannot be immediately completed cause the task that initiated the operation to block. If a socket is marked non-blocking, an operation request that cannot complete without waiting does not execute, and an error is returned.
FIOREAD	Returns the number of bytes stored in the socket's input buffer in the integer pointed to by <code>arg</code> .
FIOSETOWN	Assigns an owner to the socket. The parameter <code>arg</code> should point to an integer that provides the task ID (<code>tid</code>) of the socket's owner. This <code>tid</code> is passed as an input parameter to the user signal handler.
SIOCATMARK	Determines whether out-of-band is available. If the data available at the socket is out-of-band data, the integer pointed to by <code>arg</code> is set equal to 1. Otherwise, it equals 0 upon return.
SIOCSSBMAX	Upon creation of a socket, pNA+ assigns a maximum total default size of 128 Kbytes for send and receive socket buffer queues. This operation changes the maximum total size. This may be used to increase end-to-end throughput for fast networks.
SIOCSBMAX	Gets the total maximum send and receive socket buffer queue size that pNA+ assigns to newly created sockets.

NI-Related Operations

The following operations are available to access or modify the characteristics of a Network Interface:

Operation	Description
SIOCSIFADDR	Sets the interface address.
SIOCGIFADDR	Gets the interface address.
SIOCSIFBRDADDR	Sets the IP broadcast address of the NI.
SIOCGIFBRDADDR	Gets the IP broadcast address of the NI.

Operation	Description
SIOCSIFDSTADDR	Sets point-to-point address for the interface.
SIOCGIFDSTADDR	Gets point-to-point address for the interface.
SIOCSIFMTU	Sets the maximum transmission unit of the NI.
SIOCGIFMTU	Gets the maximum transmission unit of the NI.
SIOCSIFNETMASK	Sets the network mask.
SIOCGIFNETMASK	Gets the network mask.
SIOCSIFFLAGS	Sets the interface flags field. If the interface is marked down, any packets currently routed through the interface are re-routed or dropped, resulting in a send error condition. IFF_POLL, IFF_EXTLOOPBACK and IFF_UP flags can be set by using this call.
SIOCGIFFLAGS	Gets interface flags.
SIOCGIFCONF	Gets the interface configuration list. When this command is used, <code>arg</code> must point to an <code>ifconf</code> structure (see below). The <code>ifc_len</code> field initially should be set to the buffer size pointed to by <code>ifc_buf</code> . On return, <code>ifc_len</code> has the configuration list length in bytes.

```

/*
 * Structure used in SIOCGIFCONF request.
 * Used to retrieve interface configuration
 * for machine (useful for programs that must
 * know all accessible networks.)
 */
struct ifconf {
    int ifc_len;          /* size of associated buffer */
    union {
        char *ifcu_buf;
        struct ifreq *ifcu_req;
    } ifc_ifcu;
#define ifc_buf ifc_ifcu.ifcu_buf /* buffer address */
#define ifc_req ifc_ifcu.ifcu_req /* array of structures returned */
};

```

For all other NI-related operations, `arg` must point to the following structure:

```

struct ifreq {
    long ifr_ifno;        /* Interface number of the NI */
    union {
        struct sockaddr ifru_addr;          /* IP address of the NI */
        struct sockaddr ifru_dstaddr;       /* Dest addr p-to-p link */
        struct sockaddr ifru_broadaddr;     /* NI broadcast address */
        unsigned long ifru_flags;          /* Flags for the NI */
        int ifru_mtu;                      /* Maximum number of

```

```
        * transmission units */
        char *ifru_data; /* For private use */
    } ifr_ifru;
#define ifr_addr    ifr_ifru.ifru_addr        /* Address */
#define ifr_dstaddr ifr_ifru.ifru_dstaddr     /* Other end of */
                                                /* p-to-p link */
#define ifr_broadaddr ifr_ifru.ifru_broadaddr /*NI broadcast addr.*/
#define ifr_mtu ifr_ifru.ifru_mtu /* Maximum number of
                                    * transmission units */
#define ifr_data ifr_ifru.ifru_data /* NI private data */
#define ifr_flags ifr_ifru.ifru_flags /* Flags */
};
```

ifr_flags can contain one or more of the symbolic constants below (defined in pna.h), in the following syntax:

```
IFF_BROADCAST | IFF_RAWMEM
```

Symbolic Constant	Description
IFF_BROADCAST	NI can broadcast.
IFF_EXTLOOPBACK	NI uses external loopback.
IFF_INITDOWN	Initialize an interface in the DOWN state. By default inter- faces are installed in the UP state. This flag may only be specified when initially adding an interface. Note that this is not an attribute of the NI.
IFF_MULTICAST	NI supports multicast.
IFF_NOARP	NI does not have ARP.
IFF_POINTTTOPOINT	NI is point-to-point driver.
IFF_POLL	pNA+ polls NI at regular intervals.
IFF_RAWMEM	NI passes packets as mblks.
IFF_UNNUMBERED	NI is an unnumbered link.
IFF_UP	NI is UP.

Memory-Related Operations

SIOCGMBSTAT	Gets the statistics for <i>mblks</i> (memory blocks) configured in the system. A pointer to the mbstat structure is passed via the arg parameter of the ioctl() call. mb- stat is defined as follows:
-------------	--

```

struct mbstat {
    long mb_classes; /* Number of buffer classes */
    long mb_mblks;   /* Number of mblks configured */
    long mb_free;    /* Number of free mblks */
    long mb_wait;    /* Number of times task waited for mblk */
    long mb_drops;   /* Number of times mblks unavailable */
};

```

SIOCGDBSTAT

Gets the statistics for buffers configured in the system. A pointer to the `dbreq` structure is passed as the `arg` parameter. The `db` element must point to a buffer that can hold at least `size` number of bytes. The buffer on return contains a sequence of `dbstat` structures each of which is filled in the statistics of the particular buffer size. It is recommended that the size of the `db` buffer be large enough to contain all the buffer types configured in the system. `size` is an input/output element that contains the size of a buffer on input. On output the pNA+ network manager returns the size of buffer used by the call. These structures are defined as follows in `<pna.h>`:

```

struct dbreq {
    long size; /* Size of the buffer */
    struct dbstat *db; /* Pointer to the buffer */
};

struct dbstat {
    long db_size; /* Size of buffer */
    long db_nbuffers; /* Number of buffers configured */
    long db_free; /* Number of free buffers */
    long db_wait; /* No. of times tasks waited for buffer */
    long db_drops; /* No. of times buffer was unavailable */
};

```

ARP-Related Operations

The following operations are available for accessing and modifying the ARP table:

Operation	Description
SIOCSARP	Sets an ARP entry.
SIOCGRP	Gets an ARP entry.
SIOCGRP	Deletes an ARP entry.

For all ARP-related operations, `arg` must point to the following structure:

```

struct arpreq {
    struct sockaddr arp_pa;    /* protocol address */
    struct sockaddr arp_ha;    /* hardware address */
    int arp_flags;            /* flags */
};

```

The `arp_pa` structure is used to specify the host's IP address. The address family field in `arp_pa` must be `AF_INET`. The `arp_ha` structure is used to specify the host's hardware address. The `arp_ha` address field must be `AF_UNSPEC`.

`arp_flags` must be one of the following symbolic constants, defined in `pna.h`:

Symbolic Constant	Description
<code>ATF_PERM</code>	Permanent entry.
<code>ATF_PUBL</code>	Publish (respond for other host.)

4

`ATF_PERM` makes the entry permanent if the `ioctl()` call succeeds. `ATF_PUBL` specifies that the ARP protocol should respond to ARP requests coming from other machines for the indicated host. This allows a host to act as an ARP server, which might be useful in convincing an ARP-only machine to talk to a non-ARP machine.

Routing-Related Operations

The following operations are available for manipulating the Routing Table:

Operation	Description
<code>SIOCADDRT</code>	Adds a routing table entry.
<code>SIOCDELRT</code>	Deletes a routing table entry.
<code>SIOCMODRT</code>	Modifies a routing table entry.

For all Routing-related operations, `arg` must point to `struct rtenry`. If a subnet mask must be specified with the route, the `rt_flags` field must have the `RTF_MASK` flag set and the `rt_netmask` field must be filled with the subnet mask.

The interface number of the route's output interface can be specified. Usually this is computed internally. But for unnumbered point-point links, for instance, you could specify an interface. The `rt_ifno` field is ignored unless the `RTF_INTF` flag is set.

```

struct rtenry {
    struct sockaddr rt_dst;    /* Specifies the destination
                               * IP address of the route. */
    struct sockaddr rt_gateway; /* Specifies the gateway

```

```

                                * IP address for the route. */
unsigned short rt_flags;        /* Specifies the type of route*/
unsigned short reserved;        /* Reserved */
unsigned long  rt_netmask;      /* netmask of route */
long  rt_ifno;                  /* interface number */
unsigned long reserved2[2];     /* Reserved */
};
```

The `rt_flags` element can contain one or more of the symbolic constants below (defined in `pna.h`), in the following syntax:

```
RTF_MASK | RTF_GATEWAY
```

Symbolic Constant	Description
RTF_HOST	Host entry (net otherwise).
RTF_GATEWAY	Destination is a gateway.
RTF_UP	Route is usable.
RTF_DYNAMIC	Route is added dynamically using the ICMP Redirect message.
RTF_MODIFIED	Route is modified using the ICMP Redirect message.
RTF_INTF	Route includes intf num.
RTF_MASK	Route includes subnet mask.

Network Node ID Operations

The following operations are available for accessing the pNA+ Network Node ID. This is also known as the Router ID. The Network Node ID may be equal to any one of the IP addresses assigned to the node. To set or get the Network Node ID, `arg` must point to an `in_addr` structure containing the IP address.

Operation	Description
SIOCNNODEID	Sets the Network Node ID.
SIOCGNNODEID	Gets the Network Node ID.

Setting the IP TTL

Each IP packet sent by pNA+ (TCP/UDP or Raw IP) is assigned a TTL value. pNA+ assigns a default TTL value of 64 for outgoing UDP and TCP packets as per RFC 1700. Note that this system wide default value affects all sockets. The system wide

default TTL value may be accessed by the IP group MIB commands SIOCGIPDEFAULTTTL and SIOCSIPDEFAULTTTL.

ICMP and Raw IP packets are assigned a fixed default TTL value for 255. This system wide default TTL value cannot be changed.

For UDP/IP multicast packets the default TTL value is defined to be 1 but may be modified using the `setsockopt()` call.

The following operations are available to change the TTL value on a per socket/connection basis. Initially the per socket TTL is set as per the rules above. The TTL value may be changed for each socket. To set or get the IP TTL value the `arg` parameter must point to an integer. The TTL value must be non-negative.

Operation	Description
SIOCSIPTTL	Sets the IP TTL value of the socket.
SIOCGIPTTL	Gets the IP TTL value of the socket.

UDP Checksum Operations

The following operations may be used to access or modify the UDP checksum computation policy. By default, UDP checksum is not computed for outgoing UDP packets. The `arg` parameter must point to an integer. The integer is set to 1 to enable UDP checksum computation or 0 to disable the computation.

Operation	Description
SIOCSUDPCHKSUM	Sets the UDP checksum computation flag.
SIOCGUDPCHKSUM	Gets the UDP checksum computation flag.

MIB-II Related Operations

The `ioctl()` call is used to access pNA+ MIB-II objects, defined in this subsection. Refer to *pSOSystem System Concepts* for more details on set and get operations.

The operations described in the remainder of the `ioctl()` call description are defined by symbolic constants in `<pna_mib.h>`:

Definitions for Interface Group MIB Variables

GET Command Definitions

<code>SIOCGIFNUMBER</code>	Total number of interfaces
<code>SIOCGIFTABLE</code>	pNA+ Network Interface table
<code>SIOCGIFDESCR</code>	Description of NI
<code>SIOCGIFTYPE</code>	NI type
<code>SIOCGIFMTUNIT</code>	NI maximum transmission unit (mtu)
<code>SIOCGIFSPEED</code>	NI speed
<code>SIOCGIFPHYSADDRESS</code>	NI physical address
<code>SIOCGIFADMINSTATUS</code>	NI administration status
<code>SIOCGIFOPERSTATUS</code>	NI operational status
<code>SIOCGIFLASTCHANGE</code>	Last change in status of the NI
<code>SIOCGIFINOCETTS</code>	Number of octets received by the NI
<code>SIOCGIFINUCASTPKTS</code>	Number of unicast packets received by the NI
<code>SIOCGIFINNUCASTPKTS</code>	Number of multicast packets received by the NI
<code>SIOCGIFINDISCARDS</code>	Number of packets discarded by the NI
<code>SIOCGIFINERRORS</code>	Number of error packets received by the NI
<code>SIOCGIFINUNKNOWNPROTOS</code>	Number of packets discarded by the NI due to unknown protocols
<code>SIOCGIFOUTOCTETS</code>	Number of octets transmitted by the NI
<code>SIOCGIFOUTUCASTPKTS</code>	Number of unicast packets sent by the NI
<code>SIOCGIFOUTNUCASTPKTS</code>	Number of non-unicast packets sent by the NI
<code>SIOCGIFOUTDISCARDS</code>	Number of outbound packets discarded by the NI
<code>SIOCGIFOUTERRORS</code>	Number of outbound packets discarded by the NI due to errors
<code>SIOCGIFOUTQLEN</code>	Length of output packet queue of the NI
<code>SIOCGIFSPECIFIC</code>	NI-specific parameter

SET Command Definitions

<code>SIOCSIFADMINSTATUS</code>	Set interface administration status of the NI
---------------------------------	---

Definitions for IP Group MIB Variables

GET Command Definitions

SIOCGIPFORWARDING	IP gateway indication variable
SIOCGIPDEFAULTTTL	IP header default time-to-live value
SIOCGIPINRECEIVES	Input datagrams received from interfaces
SIOCGIPINHDRERRORS	Drops due to format errors
SIOCGIPINADDRERRORS	Drops due to invalid addresses
SIOCGIPFORWDATAGRAMS	IP datagrams forwarded
SIOCGIPINUNKNOWNPROTOS	IP datagrams discarded due to unknown protocol
SIOCGIPINDISCARDS	Input datagrams discarded with no problems
SIOCGIPINDELIVERS	Datagrams delivered to IP user-protocols
SIOCGIPOUTREQUESTS	Datagrams supplied by IP user-protocols
SIOCGIPOUTDISCARDS	Outbound datagrams discarded
SIOCGIPOUTNOROUTES	IP datagrams dropped due to no routes
SIOCGIPREASMTIMEOUT	IP reassembly queue timeout
SIOCGIPREASMREQDS	IP fragments needing reassembly
SIOCGIPREASMOKS	IP fragments reassembled
SIOCGIPREASMFAILS	IP fragments reassembly failures
SIOCGIPFRAGOKS	IP datagrams successfully fragmented
SIOCGIPFRAGFAILS	IP datagram fragmentation failures
SIOCGIPFRAGCREATES	IP fragments created
SIOCGIPROUTINGDISCARDS	IP Routing entities discarded

SET Command Definitions

SIOCSIPFORWARDING	IP gateway indication variable
SIOCSIPDEFAULTTTL	IP header default time-to-live value
SIOCSIPREASMTIMEOUT	IP fragmentation reassembly queue timeout

Definitions for IP NI Address Table

GET Command Definitions

<code>SIOCGIPADDRTABLE</code>	pNA+ NI IP address table
<code>SIOCGIPADENTADDR</code>	IP address of the NI
<code>SIOCGIPADENTIFINDEX</code>	Interface number of NI
<code>SIOCGIPADENTNETMASK</code>	Subnet mask of the NI
<code>SIOCGIPADENTBCASTADDR</code>	Broadcast address of the NI
<code>SIOCGIPADENTREASMMAXSIZE</code>	Maximum reassembly size of IP datagram

Definitions for IP Route Table

GET Command Definitions

<code>SIOCGIPROUTETABLE</code>	IP routing table
<code>SIOCGIPROUTEDEST</code>	Route destination IP address
<code>SIOCGIPROUTEIFINDEX</code>	Interface number of the NI for the route
<code>SIOCGIPROUTENEXTHOP</code>	IP address of next hop of this route
<code>SIOCGIPROUTETYPE</code>	Type of this route
<code>SIOCGIPROUTEPROTO</code>	Protocol used by the route
<code>SIOCGIPROUTEMASK</code>	Network mask to be ANDed with destination address

SET Command Definitions

<code>SIOCSIPROUTEDEST</code>	Route destination IP address
<code>SIOCSIPROUTENEXTHOP</code>	IP addr of next hop of this route
<code>SIOCSIPROUTETYPE</code>	Type of this route

Definitions for IP NET-TO-MEDIA Table

GET Command Definitions

<code>SIOCGIPNETTOMEDIATABLE</code>	IP Net-to-Media table
-------------------------------------	-----------------------

GET Command Definitions

SIOCGIPNETTOMEDIAIFINDEX	Network Interface number of the NI for which the entry is valid
SIOCGIPNETTOMEDIAPHYSADDRESS	Physical address of this entry
SIOCGIPNETTOMEDIANETADDRESS	IP address of this entry
SIOCGIPNETTOMEDIATYPE	Type of this entry

SET Command Definitions

SIOCSIPNETTOMEDIAPHYSADDRESS	Physical address of this entry
SIOCSIPNETTOMEDIANETADDRESS	IP address of this entry
SIOCSIPNETTOMEDIATYPE	Type of this entry

Definitions for ICMP Group MIB Variables

GET Command Definitions

SIOCGICMPINMSGs	ICMP messages received
SIOCGICMPINERRORs	ICMP messages with format errors
SIOCGICMPINDESTUNREACHs	ICMP Destination Unreachable messages received
SIOCGICMPINTIMEEXCDs	ICMP Time Exceeded messages received
SIOCGICMPINPARAMPROBs	ICMP Parameter Problem messages received
SIOCGICMPINSRCQUENCHs	ICMP Source Quench messages received
SIOCGICMPINREDIRECTs	ICMP Redirect messages received
SIOCGICMPINECHOs	ICMP Echo (request) messages received
SIOCGICMPINECHOREPs	ICMP Echo Reply messages received
SIOCGICMPINTIMESTAMPs	ICMP Timestamp (request) messages received
SIOCGICMPINTIMESTAMPREPs	ICMP Timestamp Reply messages received
SIOCGICMPINADDRMASKs	ICMP Address Mask Request messages received
SIOCGICMPINADDRMASKREPs	ICMP Address Mask Reply messages received
SIOCGICMPOUTMSGs	ICMP messages this entity sent
SIOCGICMPOUTERRORs	ICMP messages not sent due to ICMP problems
SIOCGICMPOUTDESTUNREACHs	ICMP Destination Unreachable messages sent
SIOCGICMPOUTTIMEEXCDs	ICMP Time Exceeded messages sent
SIOCGICMPOUTPARAMPROBs	ICMP Parameter Problem messages sent
SIOCGICMPOUTSRCQUENCHs	ICMP Source Quench messages sent
SIOCGICMPOUTREDIRECTs	ICMP Redirect messages sent
SIOCGICMPOUTECHOs	ICMP Echo (request) messages sent
SIOCGICMPOUTECHOREPs	ICMP Echo Reply messages sent

GET Command Definitions

SIOCGICMPOUTTIMESTAMPS	ICMP Timestamp (request) messages sent
SIOCGICMPOUTTIMESTAMPREPS	ICMP Timestamp Reply messages sent
SIOCGICMPOUTADDRMASKS	ICMP Address Mask Request messages sent
SIOCGICMPOUTADDRMASKREPS	ICMP Address Mask Reply messages sent

Definitions for TCP Group MIB Variables

GET Command Definitions

<code>SIOCGTCPRTOALGORITHM</code>	TCP retransmission algorithm
<code>SIOCGTCPRTOMIN</code>	TCP minimum retransmission timeout
<code>SIOCGTCPRTOMAX</code>	TCP maximum retransmission timeout
<code>SIOCGTCPMAXCONN</code>	TCP maximum simultaneous connections
<code>SIOCGTCPACTIVEOPENS</code>	Number of direct transitions to SYN-SENT state from the CLOSED state
<code>SIOCGTCPPASSIVEOPENS</code>	Number of direct transitions to SYN-RCVD state from the LISTEN state
<code>SIOCGTCPATTEMPTFAILS</code>	Number of failed TCP connection attempts
<code>SIOCGTCPESTABRESETS</code>	Number of TCP connections reset
<code>SIOCGTCPCURRENTTAB</code>	Number of current TCP connections
<code>SIOCGTCPINSEGS</code>	Number of TCP segments received
<code>SIOCGTCPOUTSEGS</code>	Number of TCP segments sent
<code>SIOCGTCPRETRANSEGS</code>	Number of TCP segments retransmitted
<code>SIOCGTCPCONNTABLE</code>	TCP connection table
<code>SIOCGTCPCONNSTATE</code>	State of this TCP connection
<code>SIOCGTCPINERRS</code>	Number of TCP segments received in error
<code>SIOCGTCPOUTRSTS</code>	Number of TCP segments sent with RST flag

SET Command Definitions

<code>SIOCSTCPCONNSTATE</code>	State of this TCP connection
--------------------------------	------------------------------

Definitions for UDP MIB Variables

GET Command Definitions

<code>SIOCGUDPINDATAGRAMS</code>	UDP datagrams delivered to UDP users
<code>SIOCGUDPNOPORTS</code>	UDP datagrams received for unknown ports

GET Command Definitions

SIOCGUDPINERRORS	UDP datagrams received with other errors
SIOCGUDPOUTDATAGRAMS	UDP datagrams sent from this entity
SIOCGUDPTABLE	pNA+ UDP listener table

Return Value

This system call returns 0 if successful, otherwise it returns -1.

Error Codes

Hex	Mnemonic	Description
0x5006	ENXIO	No such address
0x5009	EBADS	Invalid socket descriptor
0x5011	EEXIST	Requested to duplicate an existing entry
0x5016	EINVALID	Invalid argument
0x502D	EOPNOTSUPP	Requested operation not valid for this type of socket
0x504B	ETID	Invalid task ID
0x5037	ENOBUFS	Insufficient resources available to install a new route

See Also

socket, setsockopt, getsockopt

listen

Listens for connections on a socket.

```
#include <pna.h>
long listen(
    int s,          /* socket descriptor */
    int backlog     /* packet queue depth */
)
```

Description

This system call sets up the specified socket to receive connections. Connection requests are queued on the socket until they are accepted with the `accept()` call. The maximum length of the queue of pending connections must be specified. If a connection request arrives while the queue is full, the requesting client gets an `ECONNREFUSED` error.

Arguments

<code>s</code>	Specifies the socket.
<code>backlog</code>	Defines the maximum length of the queue of pending connections.

Return Value

This system call returns 0 if successful, otherwise it returns -1.

Error Codes

Hex	Mnemonic	Description
0x5016	<code>EINVAL</code>	An argument is invalid.
0x502D	<code>EOPNOTSUPP</code>	The requested operation isn't valid for this socket type.

See Also

`accept`, `connect`, `socket`

pna_allocb Allocates a message block.

```
#include <pna.h>
mblk_t *pna_allocb(size, pri)
int size;
int pri;
```

Description

`pna_allocb` allocates a message block with a data buffer of a specified size.

The pNA+ memory manager searches the buffer list for the size that best fits the requested size. If a buffer of that size is not available, the call returns `NULL`. `pna_allocb` uses the following algorithm to find the best fit:

1. The pNA+ memory manager first searches for an exact match.
2. If a match is not available, the pNA+ memory manager searches for the smallest size able to contain the requested size.
3. If none is available, the maximum size configured in the pNA+ memory manager is used.

The following example illustrates this algorithm:

Let buffers of sizes 0, 128, 1024, and 4096 bytes be configured in pNA+. If a buffer of size 1024 is requested, the pNA+ memory manager allocates a buffer from the 1024-byte buffer list. A request for a 2048-byte buffer results in the pNA+ memory manager allocating a buffer from the 4096-byte buffer list, and a request for a size 8192 buffer results in the allocation of a 4096-byte buffer.

Arguments

<code>size</code>	Specifies the size of the data buffer.
<code>pri</code>	Unused by the pNA+ memory manager.

Return Value

This system call returns a pointer to the message block if successful; otherwise, it returns a null pointer.

Error Codes

None.

See Also

`pna_esballoc`, `pna_freeb`, `pna_freemsg`

pna_esballoc Attaches a message block to the data buffer.

```
#include <pna.h>
mblk_t *pna_esballoc(buffer, size, pri, frtn)
unsigned char *buffer;
int size;
int pri;
frtn_t *frtn;
```

4

Description

`pna_esballoc` allocates and attaches a message block to the user-supplied data buffer; it uses a zero-sized data block to attach the message block to the data buffer.

Arguments

<code>buffer</code>	Points to the user-supplied data buffer.
<code>size</code>	Specifies the size of <code>buffer</code> .
<code>pri</code>	Unused by the pNA+ memory manager.
<code>frtn</code>	Points to the <code>free_rtn</code> structure, which specifies a free routine and an argument to the free routine. The free routine is called by the pNA+ memory manager when the user-specified data buffer is being freed. The <code>free_rtn</code> structure is defined in <code><pna.h></code> as follows:

```
struct free_rtn {
    void (*free_func)(); /* User free routine */
    void *free_arg;      /* Argument to free routine */
};
typedef struct free_rtn frtn_t;
```

Return Value

This system call returns a pointer to the message block if successful; otherwise, it returns a null pointer.

Error Codes

None.

See Also

`pna_allocb`, `pna_freeb`, `pna_freemsg`

pna_freeb Frees a message block.

```
#include <pna.h>
void pna_freeb(bp)
mblk_t *bp;
```

Description

`pna_freeb()` deallocates a specified message block. This function decrements the reference to the data buffer. It then deallocates the data block and the associated data buffer when no more references to them exist. If the data buffer is user-supplied [see `pna_esballoc()`], the user-supplied free routine is called when no more references to the data buffer exist.

4

Arguments

`bp` Points to the message block to be deallocated.

Return Value

None.

Error Codes

None.

See Also

`pna_allocb`, `pna_esballoc`, `pna_freemsg`

pna_freemsg Frees all message blocks associated with a message.

```
#include <pna.h>
void pna_freemsg(bp)
mblk_t *bp;
```

Description

`pna_freeb` frees all the message blocks and data blocks associated with the specified message. This routine calls the `pna_freeb()` routine to free individual message blocks.

Arguments

`bp` Points to the message to be freed.

Return Value

None.

Error Codes

None.

See Also

`pna_allocb`, `pna_esballoc`, `pna_freeb`

recv Receives data from a socket.

```
#include <pna.h>
long recv(
    int s,          /* socket descriptor */
    char *buf,      /* packet */
    int len,        /* packet length */
    int flags       /* packet attributes */
)
```

4

Description

The `recv()` system call is used to receive data from the specified socket. The behavior of this system call depends on the socket type, as described under “Arguments.”

The `recv()` system call returns the number of bytes received, and this value should always be checked because this is the only way to detect the actual number of data bytes stored in the user buffer.

Applications can use this call to receive messages from the pNA+ network manager in a linked list of `mblks` (message blocks) by setting the `MSG_RAWMEM` flag. Using `mblks` eliminates the data copy performed in the pNA+ network manager during the data transfer between the application and the pNA+ network manager.

Arguments

s Specifies the socket from which data is received. `s` can be a stream, a datagram, or a raw socket.

If `s` is a stream socket, `recv()` copies whatever data is available at the socket to the user buffer and returns. `recv()` never copies more than `len` bytes of data to the user buffer, but it can copy less, if less than `len` bytes are available. Unless `ioctl()` was used to mark the socket non-blocking, `recv()` blocks the caller if no data is available at the socket. The caller is unblocked when data is received. If the socket has been marked non-blocking, `recv()` returns immediately whether or not data is received.

If `s` is a datagram socket, every `recv()` call receives one datagram. The sender defines the size of the datagram. If the `len` parameter is less than the size of the datagram, part of the datagram is discarded. The next `recv()` call reads the next datagram received but not the unread part of the previous datagram. Unless `ioctl()` was used to mark the socket non-blocking, `recv()` blocks the caller until a datagram is available at the socket. If the socket has been marked non-blocking, `recv()` returns immediately whether or not datagrams are received.

If `s` is a raw socket, every `recv()` call receives one raw datagram. The size of the raw datagram is defined by the sender. If the `len` parameter is less than the size of the raw datagram, part of the raw datagram is discarded. The next `recv()` call reads the next raw datagram received, not the unread part of the previous raw datagram. Unless `ioctl()` was used to mark the socket non-blocking, `recv()` blocks the caller until a raw datagram is available at the socket. If the socket has been marked non-blocking, `recv()` returns immediately whether or not raw datagrams are received. The packet contains an IP header along with the packet body, if any.

<code>buf</code>	Points to the user buffer where the data is stored.
<code>len</code>	Specifies the size in bytes of the buffer.
<code>flags</code>	Specifies usage options and is the result of an OR operation performed on one or more of the following symbolic constants (defined in <code><pna.h></code>). Can also be set to 0.
<code>MSG_OOB</code>	Specifies that you want <code>recv()</code> to read any out-of-band data present on the socket, rather than the regular in-band data.
<code>MSG_PEEK</code>	Specifies that you want <code>recv()</code> to peek at the data present on the socket; the data is returned, but not consumed, so that a subsequent receive operation sees the same data.
<code>MSG_RAWMEM</code>	Specifies that you have set <code>buf</code> to point to a linked list of mblks and <code>len</code> to the total size of the message.

Return Value

This system call returns either the number of bytes received or -1 if an error occurs. When the receive is shutdown by either end of the connection, a value of 0 is returned.

Error Codes

Hex	Mnemonic	Description
0x5009	EBADS	The socket descriptor is invalid.
0x5016	EINVALID	An argument is invalid.
0x5023	EWOULDBLOCK	This operation would block and the socket is marked non-blocking.
0x5036	ECONNRESET	The connection has been reset by the peer.
0x5037	ENOBUFS	An internal buffer is required but cannot be allocated.
0x5039	ENOTCONN	The socket is not connected.

See Also

connect, recvfrom, recvmsg, socket

recvfrom Receives data from a socket.

```
#include <pna.h>
long recvfrom(
    int s,           /* socket descriptor */
    char *buf,       /* packet buffer */
    int len,         /* packet buffer length */
    int flags,       /* packet attributes */
    struct sockaddr_in *from, /* sender attributes */
    int *fromlen      /* number of bytes recieved */
)
```

Description

The `recvfrom()` system call is used to receive data from a socket. This system call is almost identical to `recv()`. The difference is `recvfrom()` may also return the address of the sender in the specified parameter.

Arguments

<code>s</code>	Specifies the socket from which data is received. The behavior of the system call depends on the socket type. Refer to <code>recv()</code> for more information.
<code>buf</code>	Points to the user buffer where data is stored.
<code>len</code>	Specifies the size of the buffer in bytes.
<code>flags</code>	Specifies usage options and is the result of an OR operation performed on one or more of the following symbolic constants (defined in <code><pna.h></code>). Can also be set to 0.
<code>MSG_OOB</code>	Specifies that you want <code>recvfrom()</code> to read any out-of-band data present on the socket, rather than the regular in-band data.
<code>MSG_PEEK</code>	Specifies that you want <code>recvfrom()</code> to peek at the data present on the socket; the data is returned, but not consumed, so that a subsequent receive operation sees the same data.

MSG_RAWMEM Specifies that you have set `buf` to point to a linked list of `mblks` and `len` to the total size of the message.

MSG_INTERFACE Specifies that you want the interface number of the NI on which the packet arrived to be stored in `from`.

If `from` is not a NULL pointer, `recvfrom()` fills in the `sockaddr_in` structure it points to with the address of the received data's sender.

The structure `sockaddr_in` is defined in `<pna.h>` and has the following format:

```
struct sockaddr_in {
    short sin_family;           /* must be AF_INET */
    unsigned short sin_port;    /* 16-bit port number */
    struct in_addr sin_addr;    /* 32-bit IP address */
    char sin_zero[8];          /* must be 0 */
};
```

This structure cannot be packed.

If `flags` includes the `MSG_INTERFACE` constant, then the structure pointed to by `from` is filled with the structure `sockaddr_intf`. This is supported only for datagram sockets. This feature is useful with unnumbered links where it may not be clear which interface the packet arrived on.

The structure `sockaddr_intf` is defined in `<pna.h>` and has the following format:

```
struct sockaddr_intf {
    short sin_family;           /* must be AF_INET */
    unsigned short sin_port;    /* 16-bit port number */
    struct in_addr sin_addr;    /* 32-bit IP address */
    long sin_ifno;              /* 32-bit interface number */
    char sin_zero[4];          /* must be 0 */
};
```

The field `sin_ifno` identifies the interface number of the incoming message's receiving interface.

`fromlen` An input-output parameter. On input, it should point to an integer equal to 16, which is the size in bytes of both `struct sockaddr_in` and `struct sockaddr_intf`.

Return Value

This system call returns either the number of bytes received or -1 if an error occurred. When the receive is shutdown by either end of the connection, a value of 0 is returned.

Error Codes

Hex	Mnemonic	Description
0x5009	EBADS	The socket descriptor is invalid.
0x5016	EINVALID	An argument is invalid.
0x5023	EWOULDBLOCK	This operation would block and the socket is marked nonblocking.
0x5036	ECONNRESET	The connection has been reset by the peer.
0x5037	ENOBUFS	An internal buffer is required but cannot be allocated.
0x5039	ENOTCONN	The socket is not connected.

See Also

`recv`, `recvmsg`, `socket`

recvmsg Receives data from a socket.

```
#include <pna.h>
long recvmsg(
    int s,                /* socket descriptor */
    struct msghdr *msg,    /* packet */
    int flags              /* packet attributes */
)
```

4

Description

The `recvmsg()` system call is used to receive data from a socket. This system call is similar to `recvfrom()` but requires fewer input parameters. Refer also to `recv()` for more details. Note that the `MSG_RAWMEM` option is not supported by this call.

Arguments

s Specifies the socket from which data is received. The behavior of the system call depends on the socket type. Refer to `recv()` for more information.

msg Points to the structure `msghdr`, which is defined in the file `<pna.h>` with the following format:

```
struct msghdr {
    char *msg_name;          /* optional address */
    int msg_namelen;         /* size of address */
    struct iovec *msg_iov;    /* scatter/gather array */
    int msg_iovlen;          /* # elements in msg_iov */
    char *msg_accrights;      /* access rights */
    int msg_accrightslen;     /* size of access rights buffer */
};
```

This structure cannot be packed. The contents of the `msghdr` fields are described below.

msg_name If the socket is unconnected, can specify the source from which it receives data.

msg_namelen Specifies the length of the buffer pointed to by `msg_name`.

<code>msg_iov</code>	<p>Points to an array whose members (<code>msg_iov[0]</code>, ..<code>msg_iov[msglen-1]</code>) specify the buffers in which the received data is stored. The <code>iovec</code> structure has the following format:</p> <pre>struct iovec { char *iov_base: /* base address */ int iov_len; /* buffer length */ };</pre> <p>This structure cannot be packed. Each <code>iovec</code> entry specifies the base address and length of an area in memory where data is stored. <code>recvmsg()</code> always fills an area completely before it goes to the next area.</p>
<code>msg_accrights</code>	Points to a buffer that receives the access rights information sent along with a message. This applies to messages that a UNIX host sends.
<code>msg_accrightslen</code>	Specifies the length of the buffer pointed to by <code>msg_accrights</code> .
<code>flags</code>	Specifies usage options and is the result of an OR operation performed on one or more of the following symbolic constants (defined in <code><pna.h></code>). It can also be set to 0.
<code>MSG_OOB</code>	Specifies that you want <code>recvmsg()</code> to read any out-of-band data present on the socket, rather than the regular in-band data.
<code>MSG_PEEK</code>	Specifies that you want <code>recvmsg()</code> to peek at the data present on the socket; the data is returned, but not consumed, so that a subsequent receive operation sees the same data.

Return Value

This system call returns the number of bytes received, or it returns -1 if an error occurs. When the receive is shutdown by either end of the connection, a value of 0 is returned.

Error Codes

Hex	Mnemonic	Description
0x5009	EBADS	The socket descriptor is invalid.
0x5016	EINVALID	An argument is invalid.
0x5023	EWOULDBLOCK	This operation would block and the socket is marked non-blocking.
0x5028	EMSGSIZE	Message too long.
0x5036	ECONNRESET	The connection has been reset by the peer.
0x5037	ENOBUFS	An internal buffer is required but cannot be allocated.
0x5039	ENOTCONN	The socket is not connected.

See Also

recv, recvfrom, socket

select

Checks the status of multiple sockets.

```
#include <pna.h>
long select(
    int width,                /* largest descriptor list */
    fd_set *readset,          /* read descriptor list */
    fd_set *writeset,         /* write descriptor list */
    fd_set *exceptset,        /* exception list */
    struct timeval *timeout    /* timeout for operation */
)
```

Description

This system call is used to multiplex I/O requests among multiple sockets. Three sets of socket descriptors may be specified: a set of sockets from which to read, a set to which to write and a set that may have pending exceptional conditions.

Each set is actually a structure containing an array of long integer bit masks. The size of the array is set by the definition of `FD_SETSIZE` (in `<pna.h>`.) The array is long enough to hold one bit for each `FD_SETSIZE` socket descriptor.

If `select()` returns successfully, the three sets indicate which socket descriptors can be read, which can be written to, or which have exceptional conditions pending. A timeout value may be specified.

Arguments

<code>width</code>	Specifies the largest descriptor list given by <code>readset</code> , <code>writeset</code> , or <code>exceptset</code> .
<code>readset</code>	Points to a set of sockets from which to read.
<code>writeset</code>	Points to a set of sockets to which to write.
<code>exceptset</code>	Points to a set of sockets that may have an exceptional condition pending.

timeout Specifies a timeout option. If the fields in **timeout** are set to 0, **select()** returns immediately. If the **timeout** is a null pointer, **select()** blocks until a descriptor is selectable. The structure **timeval** is defined in the file `<pna.h>` and has the following format:

```
struct timeval {
    long tv_sec;    /* number of seconds */
    long tv_usec;  /* number of microseconds */
}
```

Usage

Macros

The status of a socket descriptor in a **select** mask can be tested with the **FD_ISSET(s, &mask)** macro, which returns a non-zero value if **s** is a member of the set **mask**, and 0 if it is not.

In addition, the macros **FD_SET(s, &mask)** and **FD_CLEAR(s, &mask)** are provided for adding and removing socket descriptors to and from a set. **s** is the socket descriptor, and **mask** points to a bit mask data structure. The macro **FD_ZERO(&mask)** is provided to clear the set and should be used before the set is used.

These macros are defined in the file `<pna.h>`.

Example

The following example shows how to use **select()** to determine if two sockets have available data:

```
fd_set read_mask;

struct timeval wait;

for (;;)
{
    wait.tv_sec = 1;        /* wait for 1 second */
    wait.tv_usec = 0;

    FD_ZERO (&read_mask);
    FD_SET (s1, &read_mask);
    FD_SET (s2, &read_mask);

    nb = select (FD_SETSIZE, &read_mask, (fd_set *) 0,
                (fd_set *) 0, &wait);
    if (nb <= 0)
    {
        /* error occurred or timed out */
    }
}
```

```
    }

    if (FD_ISSET(s1, &read_mask))
    {
        /* socket 1 has data available */
    }

    if (FD_ISSET(s2, &read_mask))
    {
        /* socket 2 has data available */
    }
}
```

If two tasks attempt to use `select()` on the same socket for the same conditions, an error occurs.

Return Value

A -1 is returned if an error occurs and the socket descriptor masks remain unchanged; a 0 is returned if a timeout occurs. On success a nonzero value is returned that indicates the number of descriptors on which events have occurred.

Error Codes

Hex	Mnemonic	Description
0x5009	EBADS	The socket descriptor is invalid.
0x504A	ECOLL	Collision in <code>select()</code> ; these conditions have already been selected by another task.

See Also

`accept`, `connect`, `recv`, `send`

send Sends data to a socket.

```
#include <pna.h>
long send(
    int s,      /* socket descriptor */
    char *buf,   /* packet */
    int len,     /* packet length */
    int flags    /* packet attributes */
)
```

4

Description

The `send()` system call is used to send data to a foreign socket.

If no buffer space is available at the socket to hold the data to be transmitted, `send()` blocks the calling task unless the socket has been marked non-blocking.

Applications can use this call to pass messages to the pNA+ network manager in a linked list of *mblys* (message blocks) by setting the `MSG_RAWMEM` flag (see “Arguments,” below). Using *mblys* eliminates the data copy performed in the pNA+ network manager during the data transfer between the application and the pNA+ network manager.

Arguments

<code>s</code>	<p>Specifies the local socket, which must be in a connected state.</p> <p>If <code>s</code> is a stream socket, the data is sent to the foreign socket that is connected to <code>s</code>.</p> <p>If <code>s</code> is a datagram socket, the data is sent to the socket that has been associated with <code>s</code> through a previous <code>connect()</code> system call.</p> <p>If <code>s</code> is a raw socket, the raw datagram is sent to the raw socket that has been associated with <code>s</code> through a previous <code>connect()</code> system call.</p>
<code>buf</code>	Points to a buffer containing the data to send. If <code>s</code> is a datagram socket, the data that <code>buf</code> points to is a datagram.
<code>len</code>	Specifies the number of bytes in <code>buf</code> .

flags	Specifies usage options and is the result of an OR operation performed on one or more of the following symbolic constants (defined in <code><pna.h></code>). It can also be set to 0.
MSG_OOB	Specifies that you want <code>send()</code> to send out-of-band data, rather than the regular in-band data.
MSG_DONTROUTE	Specifies that you want <code>send()</code> to turn on the socket flag <code>SO_DONTROUTE</code> for the duration of the send operation. The <code>SO_DONTROUTE</code> flag prohibits routing of outgoing data from the socket. Packets directed at unconnected nodes are dropped.
MSG_RAWMEM	Specifies that you have set <code>buf</code> to point to a linked list of mblks and <code>len</code> to the total size of the message.

Return Value

This system call returns the number of bytes sent or -1 if an error occurs.

Error Codes

Hex	Mnemonic	Description
0x5009	EBADS	The socket descriptor is invalid.
0x500D	EACCESS	The broadcast option is not set for this socket.
0x5016	EINVALID	An argument is invalid.
0x5020	EPIPE	The connection is broken.
0x5023	EWouldBlock	This operation would block (and the socket is marked non-blocking.)
0x5028	EMSGSIZE	Message too long.
0x5033	ENETUNREACH	Destination network can't be reached from this node.
0x5036	ECONNRESET	The connection has been reset by the peer.
0x5037	ENOBUFS	An internal buffer is required but cannot be allocated.
0x5039	ENOTCONN	The socket is not connected.

Hex	Mnemonic	Description
0x5041	EHOSTUNREACH	The destination host could not be reached from this node.

See Also

sendto, sendmsg, socket

sendmsg Sends data to a socket.

```
#include <pna.h>
long sendmsg(
    int s,                /* socket descriptor */
    struct msghdr *msg,    /* packet structure */
    int flags              /* packet attributes */
)
```

Description

The `sendmsg()` system call is used to send data to a foreign socket. This system call is similar to `sendto()`, but it requires fewer input parameters and uses the structure `msghdr`. For a complete description of this system call, refer to the `sendto` call description on page 4-62.

Note that the `MSG_RAWMEM` option is not supported by this call.

Arguments

<code>s</code>	Specifies the local datagram socket.
<code>msg</code>	Points to a <code>msghdr</code> structure. The <code>msghdr</code> structure is described in the <code>recvmsg</code> call description on page 4-51.
<code>flags</code>	Specifies usage options and is formed by ORing one or more of the following symbolic constants (defined in <code><pna.h></code>). It can also be set to 0.
<code>MSG_OOB</code>	Specifies that you want <code>sendmsg()</code> to send out-of-band data, rather than the regular in-band data.
<code>MSG_DONTROUTE</code>	Specifies that you want <code>sendmsg()</code> to turn on the socket flag <code>SO_DONTROUTE</code> for the duration of the operation. The <code>SO_DONTROUTE</code> flag prohibits routing of outgoing data from the socket. Packets directed at unconnected nodes are dropped.

Return Value

This system call returns the number of bytes sent or -1 if an error occurred.

Error Codes

Hex	Mnemonic	Description
0x5009	EBADS	The socket descriptor is invalid.
0x500D	EACCESS	The broadcast option is not set for this socket.
0x5016	EINVALID	An argument is invalid.
0x5020	EPIPE	The connection is broken.
0x5023	EWouldBlock	This operation would block and the socket is marked non-blocking.
0x5027	EDESTADDRREQ	The destination address is invalid.
0x5028	EMSGSIZE	The data cannot be transmitted as a unit.
0x5031	EADDRNOTAVAIL	The specified address is not available.
0x5033	ENETUNREACH	The destination network cannot be reached from this node.
0x5036	ECONNRESET	The connection has been reset by the peer.
0x5037	ENOBUFS	An internal buffer is required, but can't be allocated.
0x5038	EISCONN	The socket is in a connected state.
0x5039	ENOTCONN	The socket is not connected.
0x5041	EHOSTUNREACH	The destination host could not be reached from this node.

See Also

send, sendto, socket

sendto Sends data to a socket.

```
#include <pna.h>
long sendto(
    int s,                /* socket descriptor */
    char *buf,            /* packet */
    int len,              /* packet length */
    int flags,            /* packet attribute */
    struct sockaddr_in *to, /* destination socket type */
    int tolen             /* size of sockaddr_in */
)
```

Description

The `sendto()` system call is used to send data to a foreign datagram socket. Although it is possible to use this system call with stream, datagram, or raw sockets, it is intended to be used only with datagram sockets or raw sockets.

If no buffer space is available at the socket to hold the datagram, `sendto()` blocks the calling task unless the socket has been marked non-blocking.

Applications can use this call to pass messages to the pNA+ network manager in a linked list of `mblocks` (message blocks) by setting the `MSG_RAWMEM` flag. Using `mblocks` eliminates the data copy performed in the pNA+ network manager during the data transfer between the application and the pNA+ network manager.

Arguments

<code>s</code>	Specifies the local socket.
<code>buf</code>	Points to a buffer that contains the data to send. The data pointed to by <code>buf</code> is called a datagram.
<code>len</code>	Specifies the number of bytes in <code>buf</code> .
<code>flags</code>	Specifies usage options and is the result of an OR operation performed on one or more of the following symbolic constants (defined in <code><pna.h></code>). It can also be set to 0.
<code>MSG_OOB</code>	Specifies that you want <code>sendto()</code> to send out-of-band data, rather than the regular in-band data.

MSG_DONTROUTE	Specifies that you want <code>sendto()</code> to turn on the socket flag <code>SO_DONTROUTE</code> for the duration of the send operation. The <code>SO_DONTROUTE</code> flag prohibits routing of outgoing data from the socket. Packets directed at unconnected nodes are dropped.
MSG_RAWMEM	Specifies that you have set <code>buf</code> to point to a linked list of <code>mblks</code> and <code>len</code> to the total size of the message.
MSG_INTERFACE	Specifies the outgoing interface of the message. If no route is found to the destination, then the interface number is specified in the argument <code>to</code> , which is a pointer to the structure <code>sockaddr_intf</code> . This is supported only for datagram or raw sockets. This is helpful with unnumbered links where there may not be a route to the destination.

`to` Specifies the destination socket address and is a pointer to either the `sockaddr_in` structure or the `sockaddr_intf` structure. `sockaddr_intf` is used *only* if the `MSG_INTERFACE` option is set in flags. These structures are defined in `<pna.h>` and have the following format:

```

struct sockaddr_in {
    short sin_family;           /* must be AF_INET */
    unsigned short sin_port;    /* 16-bit port number */
    struct in_addr sin_addr;    /* 32-bit IP address */
    char sin_zero[8];          /* must be 0 */
};

struct sockaddr_intf {
    short sin_family;           /* must be AF_INET */
    unsigned short sin_port;    /* 16-bit port number */
    struct in_addr sin_addr;    /* 32-bit IP address */
    long sin_ifno;              /* 32-bit interface number */
    char sin_zero[4];          /* must be 0 */
};

```

The field `sin_ifno` identifies the interface number of the outgoing packet's output interface.

The above structures cannot be packed.

`tolen` Specifies the size in bytes of either `struct sockaddr_in` or `struct sockaddr_intf` and must be 16.

Return Value

This system call returns the number of bytes sent and -1 if an error occurs.

Error Codes

Hex	Mnemonic	Description
0x5009	EBADS	The socket descriptor is invalid.
0x500D	EACCESS	The broadcast option is not set for this socket.
0x5016	EINVALID	An argument is invalid.
0x5020	EPIPE	The connection is broken.
0x5023	EWOULDBLOCK	This operation would block, and the socket is marked non-blocking.
0x5027	EDESTADDRREQ	The destination address is invalid.
0x5028	EMSGSIZE	Message too long.
0x5031	EADDRNOTAVAIL	The specified address is not available.
0x5033	ENETUNREACH	The destination network cannot be reached from this node.
0x5036	ECONNRESET	The connection has been reset by the peer.
0x5037	ENOBUFS	An internal buffer is required but can't be allocated.
0x5038	EISCONN	The socket is in a connected state.
0x5039	ENOTCONN	The socket is not connected.
0x5041	EHOSTUNREACH	The destination host could not be reached from this node.

See Also

`send`, `sendmsg`, `select`, `socket`

set_id Sets a task's user ID and group ID.

```
#include <pna.h>
long set_id(
    long userid,    /* user identity */
    long groupid,   /* group identity */
    long *groups    /* must be zero */
)
```

4

Description

This system call sets the user ID and group ID of the calling task. These IDs are used for accessing NFS servers. Default values for the IDs are defined in the pNA+ Configuration Table.

Arguments

userid	Specifies the calling task's user ID.
groupid	Specifies the calling task's group ID.
groups	Zero must be passed as a third argument (it is currently ignored).

Return Value

This system call returns 0 if successful, otherwise it returns -1.

Error Codes

None.

See Also

get_id

setsockopt Sets options on a socket.

```
#include <pna.h>
long setsockopt(
    int s,          /* socket descriptor */
    int level,      /* SOL_SOCKET, IPPROTO_TCP */
                  /* or IPPROTO_IP */
    int optname,    /* retrieval access */
    char *optval,   /* modification data */
    int optlen      /* sizeof modification data */
)
```

Description

The `setsockopt()` system call sets options associated with the specified socket. Socket level, IP protocol level, or TCP protocol level options can be set.

Arguments

<code>s</code>	Specifies the socket whose options are to be set.
<code>level</code>	Specifies the level of the option to be set. Must be <code>SOL_SOCKET</code> for socket level operations, <code>IPPROTO_TCP</code> for TCP protocol level operations, or <code>IPPROTO_IP</code> for IP protocol level operations. These options are defined in <code><pna.h></code> .
<code>optname</code>	Specifies the option to be set and uses a symbolic constant defined in <code><pna.h></code> . The symbolic constants available for each level are described below.
<code>optval</code>	Points to a buffer in which the option's value is specified. Most options are 32-bit values. A nonzero value means the option should be set, and a 0 means the option should be turned off.
<code>optlen</code>	Specifies the size of the value pointed to by <code>optval</code> .

Socket Level Options

`level` must be set to `SOL_SOCKET` for socket level operations and the `optname` value can be one of the following (defined in `<pna.h>`):

<code>SO_BROADCAST</code>	Allows broadcast datagrams on a socket.
<code>SO_DONTROUTE</code>	Indicates that the outgoing data should not be routed. Packets directed at unconnected nodes are dropped.
<code>SO_KEEPAIVE</code>	Maintains a connection by periodically transmitting a packet over socket <code>s</code> .
<code>SO_LINGER</code>	Controls the action taken when unsent messages are queued on a socket and a <code>close()</code> is executed. If the socket is a stream socket and <code>SO_LINGER</code> is set (<code>l_onoff</code> set to 1), the calling task blocks until it is able to transmit the data or until a timeout occurs. If <code>SO_LINGER</code> is disabled (<code>l_onoff</code> set to 0), the socket is deleted immediately. <code>SO_LINGER</code> uses the <code>linger</code> structure, which is defined in <code><pna.h></code> as follows: <pre> struct linger { int l_onoff; /* on/off option */ int l_linger; /* linger time in seconds */ } </pre> <p>This structure cannot be packed.</p>
<code>SO_OOBINLINE</code>	Requests that out-of-band data be placed in the normal data input queue as it is received; it becomes accessible through <code>recv()</code> calls without the <code>MSG_OOB</code> flag.
<code>SO_RCVBUF</code>	Adjusts the normal allocated input buffer size. The buffer size can be increased for high-volume connections or decreased to limit the possible backlog of data. The pNA+ network manager limits this value to 32 Kbytes.
<code>SO_REUSEADDR</code>	Indicates that local addresses can be reused in a <code>bind()</code> call.
<code>SO_REUSEPORT</code>	Indicates that local addresses can be reused in a <code>bind()</code> call. For more information, see section 4.4.3 of the Network Programming chapter in <i>pSOSystem System Concepts</i> .
<code>SO_SNDBUF</code>	Adjusts the normal allocated output buffer size. The buffer size can be increased for high-volume connections or decreased to limit the possible backlog of data. The pNA+ network manager limits this value to 32 Kbytes.

TCP Level Option

`level` must be set to `IPPROTO_TCP` for TCP protocol level operations. The argument `optname` can have the following value (defined in `<pna.h>`):

<code>TCP_KEEPAIVE_CNT</code>	Number of Keepalive strobos. Upon expiry of the <code>Keepalive</code> idle timer TCP will send a number of strobos separated by a fixed interval. If the other end fails to respond to the strobos (special TCP segments) then the TCP connection will be terminated. The default number of strobos in pNA+ are 8. Only valid if the <code>SO_KEEPAIVE</code> option is set above.
<code>TCP_KEEPAIVE_IDLE</code>	Keepalive idle time in TCP. If the connection has been idle for this time, the timer will expire causing TCP to send a special segment forcing the other end to respond. On demand-dial links for example the timer may be set long enough so as not to cause unnecessary traffic. The default in pNA+ is 120 seconds (2 hrs). The timer is in seconds. Only valid if the <code>SO_KEEPAIVE</code> option is set above.
<code>TCP_KEEPAIVE_INTVL</code>	Keepalive strobe interval. The strobos sent out by TCP upon expiration of the <code>Keepalive</code> idle timer are separated by a fixed interval. The default interval between the strobos is set to 75 seconds in pNA+. The timer is in seconds. Only valid if the <code>SO_KEEPAIVE</code> option is set above.
<code>TCP_MSL</code>	Maximum Segment Lifetime in TCP. This controls the <code>TIME_WAIT</code> or <code>2MSL</code> timer in TCP which is set to twice the value of the MSL. The timer is used to validate connection termination and transmits remaining data in the send queue. It is a safeguard against sequence numbers being overlapped. If set to a low value it allows the sockets to be quickly deleted. The default in pNA+ is 30 seconds. The timer is in seconds.
<code>TCP_NODELAY</code>	Disables delay acknowledgment algorithm. Data is sent immediately over the network instead of waiting for the window to be full.

IP Level Options

`level` must be set to `IPPROTO_IP` for IP protocol level operations. The argument `optname` can have one of the following values (defined in `<pna.h>`):

IP_ADD_MEMBERSHIP

Join a multicast group. For this option, `optval` is a pointer to the `ip_mreq` structure (defined in `<pna.h>`). The group multicast address must be specified in the field `imr_mcastaddr`. This is a CLASS-D IP multicast address. The `imr_interface` parameter may be set to the IP address of a specific interface for which group membership will be enabled. The interface must of course support multicasting. Optionally the `imr_interface` parameter may be set to `INADDR_ANY` in which case pNA+ will decide the interface to join on (for example, if a route exists for a matching multicast address that specifies the interface). The maximum number of groups that can be joined per multicast socket is defined by the constant `IP_MAX_MEMBERSHIPS` in `pna.h`.

The structure below is defined in `<pna.h>` and used with the `IP_ADD_MEMBERSHIP` option.

```
struct ip_mreq {
    struct in_addr imr_mcastaddr;
    /* IP multicast address of group */
    struct in_addr imr_interface;
    /* local IP address of interface */
};
```

IP_ADD_MEMBERSHIP_INTF

Similar to IP_ADD_MEMBERSHIP above. The `optval` is a pointer to the structure `ip_mreq_intf` (defined in `<pna.h>`). The only difference is that the interface is defined using the interface number. If the interface number is specified as -1 then pNA+ will select the interface based upon the routing table. This option is useful for unnumbered links because the IP address of the interface is not enough to identify the interface.

The structure below is defined in `<pna.h>` and used with the IP_ADD_MEMBERSHIP_INTF option.

```
struct ip_mreq_intf {
    struct in_addr imrif_mcastaddr;
    /* IP multicast address of group */
    long imrif_ifno;
    /* local interface number */
};
```

IP_DROP_MEMBERSHIP

Leave a multicast group. `optval` is a pointer to the `ip_mreq` structure (defined in `<pna.h>`). The group multicast address to be dropped must be specified in `imr_mcastaddr`. The interface address could be set to `INADDR_ANY` unless it specifies the particular interface for which the group membership must be dropped.

The structure below is defined in `<pna.h>` and used with the IP_DROP_MEMBERSHIP option.

```
struct ip_mreq {
    struct in_addr imr_mcastaddr;
    /* IP multicast address of group */
    struct in_addr imr_interface;
    /* local IP address of interface */
};
```


IP_DROP_MEMBERSHIP_INTF Similar to **IP_DROP_MEMBERSHIP** above. The **optval** is a pointer to the structure **ip_mreq_intf** (defined in **<pna.h>**). The only difference is that the interface is defined using the interface number. If the interface number is specified as -1 then pNA+ will select the interface based upon the routing table. This option is useful for unnumbered links because the IP address of the interface is not enough to identify the interface.

The structure below is defined in **<pna.h>** and used with the **IP_DROP_MEMBERSHIP_INTF** option.

```
struct ip_mreq_intf {
    struct in_addr imrif_mcastaddr;
    /* IP multicast address of group */
    long imrif_ifno;
    /* local interface number */
};
```

IP_HDRINCL

Specifies that the IP header will be included in the output packets. The following fields will be set by pNA+ if they are set to 0 in the included IP header: IP identification number and IP source address. The fragmentation offset and checksum fields are always computed by pNA+. The rest of the IP header fields must be set appropriately.

IP_MULTICAST_IF

Specifies the outgoing interface for multicast packets. **optval** is a pointer to **struct in_addr**. If set to **INADDR_ANY** then the routing table will be used to select an appropriate interface.

IP_MULTICAST_INTF

Specifies the outgoing interface for multicast packets. The outgoing interface is defined by its interface number. The **optval** is a pointer to a long. If set to -1 then the routing table will be used to select an appropriate interface. This option is useful for unnumbered links because the IP address of the interface is not enough to identify the interface.

<code>IP_MULTICAST_LOOP</code>	Specifies whether or not to loopback multicast packets. <code>optval</code> is a pointer to an unsigned char. By default the packets are looped back (<code>IP_DEFAULT_MULTICAST_LOOP</code>). A value of 0 disables loopback.
<code>IP_MULTICAST_TTL</code>	Specifies the time-to-live for outgoing IP multicast datagrams. <code>optval</code> is a pointer to an unsigned char. The default is <code>IP_DEFAULT_MULTICAST_TTL</code> .

Return Value

This system call returns 0 if successful, otherwise it returns -1.

Error Codes

Hex	Mnemonic	Description
0x5009	EBADS	The socket descriptor is invalid.
0x5016	EINVALID	An argument is invalid.
0x502A	ENOPROTOOPT	The <code>optname</code> or <code>level</code> is not valid.
0x5030	EADDRINUSE	The multicast address is already in use.
0x5031	EADDRNOTAVAIL	The multicast address was not available because of one of the following: the multicast address was not found, the interface could not be determined, or the interface does not support multicast.
0x5037	ENOBUFS	An internal buffer is required but cannot be allocated.
0x503B	ETOOMANYREFS	Too many references; can't splice. The per socket maximum number of memberships has been exceeded. See section 3 of the <i>pSOSystem Programmer's Reference</i> .

See Also

`getsockopt`, `socket`

shr_socket Obtains a new socket descriptor for an existing socket.

```
#include <pna.h>
int shr_socket(
    int s,    /* socket descriptor */
    int tid /* task identity */
)
```

Description



This system call is used to obtain a new socket descriptor for an existing socket. The new socket descriptor can be used by the task with the specified ID to reference the socket in question.

This system call is provided for applications that implement UNIX-style server programs, which normally incorporate the UNIX `fork()` call.

Arguments

- s Specifies the existing socket descriptor to be shared.
- tid Specifies the task ID of a task that seeks to access the same socket.

Return Value

This system call returns a socket descriptor if successful, otherwise it returns -1.

Error Codes

Hex	Mnemonic	Description
0x5009	EBADS	The socket descriptor is invalid.
0x5017	ENFILE	An internal table has run out of space.
0x504B	ETID	The task ID is not valid.

shutdown

Terminates all or part of a full-duplex connection.

```
#include <pna.h>
long shutdown(
    int s,    /* socket descriptor */
    int how   /* shutdown mechanism */
)
```

Description

This system call is used to terminate all or part of a full-duplex connection on a specified socket. The socket may be shut down for sending, receiving, or both.

Arguments

s	Specifies the socket to be shut down.
how	Specifies the shutdown mechanism. Three options are available, as follows:
0	No further receives are allowed on the socket.
1	No further sends are allowed on the socket.
2	No further sends or receives are allowed on the socket.

Return Value

This system call returns 0 if successful, otherwise it returns -1.

Error Codes

Hex	Mnemonic	Description
0x5009	EBADS	The socket descriptor is not valid.
0x5016	EINVALID	An argument is not valid.
0x5039	ENOTCONN	The socket is not connected.

See Also

`connect`, `socket`

socket Creates a socket.

```
#include <pna.h>
int socket(
    int domain,    /* socket domain */
    int type,      /* socket type carrier */
    int protocol   /* socket protocol class */
)
```

Description

The `socket()` system call creates a new socket and returns its socket descriptor. The socket is an endpoint of communication.

Arguments

<code>domain</code>	Specifies the socket domain and must be set to <code>AF_INET</code> .						
<code>type</code>	Specifies one of the following types of sockets (defined in the <code><pna.h></code> file): <table border="0" style="margin-left: 20px;"> <tr> <td><code>SOCK_STREAM</code></td><td>Defines a stream socket, which uses TCP to provide a reliable connection-based communication service.</td></tr> <tr> <td><code>SOCK_DGRAM</code></td><td>Defines a datagram socket, which uses UDP to provide a datagram service.</td></tr> <tr> <td><code>SOCK_RAW</code></td><td>Defines a raw socket, which uses the protocol specified by <code>protocol</code> for a raw datagram service.</td></tr> </table>	<code>SOCK_STREAM</code>	Defines a stream socket, which uses TCP to provide a reliable connection-based communication service.	<code>SOCK_DGRAM</code>	Defines a datagram socket, which uses UDP to provide a datagram service.	<code>SOCK_RAW</code>	Defines a raw socket, which uses the protocol specified by <code>protocol</code> for a raw datagram service.
<code>SOCK_STREAM</code>	Defines a stream socket, which uses TCP to provide a reliable connection-based communication service.						
<code>SOCK_DGRAM</code>	Defines a datagram socket, which uses UDP to provide a datagram service.						
<code>SOCK_RAW</code>	Defines a raw socket, which uses the protocol specified by <code>protocol</code> for a raw datagram service.						
<code>protocol</code>	Specifies the network protocol and can be 0, TCP, UDP, or any other protocol. For raw sockets the protocol can have any value except TCP or UDP. A protocol number of zero acts as a wildcard for raw sockets, accepting any raw IP packet.						

Return Value

This system call returns a socket descriptor, or a -1 if an error occurs.

Error Codes

Hex	Mnemonic	Description
0x5016	EINVALID	An argument is invalid.
0x5017	ENFILE	An internal table has run out of space.
0x5029	EPROTOTYPE	Wrong protocol type for socket.
0x502C	EPROTONOSUPPORT	The protocol argument is not valid.
0x5037	ENOBUFS	An internal buffer is required but cannot be allocated.

See Also

accept, bind, close, connect, listen, setsockopt, getsockopt

5

pRPC+ System Calls

5

This chapter provides information on the system calls in the pRPC+ component of pSOSystem. Each call's section includes its syntax, a description, its arguments, its return value, and any error codes that it can return. Where applicable, the section also includes the headings "Notes" and "See Also." "Notes" provides any important information not specifically related to the call's description, and "See Also" indicates other calls that have related information.

If you need to look up a system call by its functionality, refer to Appendix A, "Tables of System Calls," which lists the calls alphabetically by component and provides a brief description of each call.

For more information on error codes, refer to Appendix B, "Error Codes," which lists the codes numerically and gives the pSOSystem calls that are associated with each one.

pRPC+ System Calls

The following list shows all of the services supported by the pRPC+ library. All routines implement standard ONC RPC/XDR services, except for those marked with an asterisk. The calls marked with an asterisk (*) are described in detail in this chapter.

auth_destroy	authnone_create	authunix_create
authunix_create_default	callrpc	clnt_broadcast
clnt_call	clnt_create	clnt_destroy
clnt_freeres	clnt_geterr	clnt_perrno
clnt_perror	clnt_sperrno	clnt_sperror
clnt_control	clnt_pcreateerror	clntraw_create
clnt_spcreateerror	clnt_udp_bufcreate	clntudp_create
clnttcp_create	get_fdset*	pmap_getmaps
pmap_getport	pmap_rmtcall	pmap_set
pmap_unset	registerrpc	rpc_getcreateerr*
svcerr_auth	svcerr_decode	svcerr_noproc
svcerr_noprog	svcerr_progvers	svcerr_systemerr
svcerr_weakauth	svcfid_create	svccraw_create
svctcp_create	svcudp_create	svc_destroy
svc_fdset	svc_freeargs	svc_getargs
svc_getcaller	svc_getreq	svc_getreqset
svc_run	svc_sendreply	svc_register
svc_unregister	xdrmem_create	xdrrec_create
xdrrec_endofrecord	xdrrec_eof	xdrrec_readbytes
xdrrec_skiprecord	xdrstdio_create	xdr_accepted_reply
xdr_array	xdr_authunix_parms	xdr_bool
xdr_bytes	xdr_callhdr	xdr_callmsg
xdr_char	xdr_destroy	xdr_double
xdr_enum	xdr_float	xdr_free

xdr_getpos	xdr_inline	xdr_int
xdr_long	xdr_opaque	xdr_opaque_auth
xdr_pmap	xdr_pmaplist	xdr_pointer
xdr_reference	xdr_rejected_reply	xdr_replymsg
xdr_setpos	xdr_short	xdr_string
xdr_union	xdr_u_char	xdr_u_int
xdr_u_long	xdr_u_short	xdr_void
xdr_vector	xdr_wrapstring	xprt_register
xprt_unregister		

get_fdset

```
#include <rpc.h>
void get_fdset(
    fd_set *read_mask    /* server's read file
                          /* descriptor bit mask */
)
```

Description

This pRPC+ service call provides access to the task-specific equivalent of the ONC RPC `svc_fdset` global variable.

Arguments

<code>read_mask</code>	<p>Points to the location where <code>get_fdset()</code> copies the contents of the <code>svc_fdset</code> variable. Memory pointed to by <code>read_mask</code> should be preallocated.</p> <p>The returned <code>read_mask</code> can serve as the read descriptor list argument to the pNA+ <code>select()</code> service call. The value that <code>select()</code> returns in place of the input read descriptor list can serve as an input argument to the pRPC+ <code>svc_getreqset()</code> call.</p>
------------------------	---

Return Value

None.

Error Codes

None.

See Also

The `svc_getreqset` description in other ONC RPC documentation.

rpc_getcreateerr

```
#include <rpc.h>
void rpc_getcreateerr(
    struct rpc_createerr *err    /* error buffer */
)
```

Description

This pRPC+ service call provides access to the task-specific equivalent of the ONC RPC `rpc_createerr` global variable.

Arguments

<code>rpc_createerr</code>	<p>Points to the location where <code>rpc_getcreateerr()</code> copies the contents of the <code>rpc_createerr</code> variable.</p> <p>The creation routines for the RPC client handle use <code>rpc_createerr</code> to store the reason for a creation failure. Application programs do not require access to this variable: the standard routines <code>clnt_pcreateerror()</code> and <code>clnt_spcreateerr()</code> return a textual description of the failure.</p>
----------------------------	--

Return Value

None.

Error Codes

None.

See Also

The `clnt_pcreateerror` and `clnt_spcreateerror` descriptions in other ONC RPC documentation.

6

pROBE+ and ESsp System Calls

This chapter provides detailed descriptions of the system calls supported by pROBE+ and ESsp. The calls are listed alphabetically, with a multipage section of information for each call. Each call's section includes its syntax, a detailed description, its arguments, and its return value.

The ESsp cross-system visual analyzer graphically displays your embedded application's activity on a host terminal and allows you to analyze the application's performance. ESsp accepts one system call, `log_event()`, through its target-resident application monitor, pMONT.

pROBE+ is pSOSystem's target debugger and analyzer. pROBE+ accepts two system calls: `db_input()` and `db_output()`.

If you need to look up a system call by its functionality, refer to Appendix A, "Tables of System Calls," which lists the calls alphabetically by component and provides a brief description of each call.

For more information on error codes, refer to Appendix B, "Error Codes," which lists the codes numerically and gives the pSOSystem calls that are associated with each one.

db_input Prompts and gets input from the high-level debugger.

```
long db_input (
    char *inbuf,           /* user input buffer */
    unsigned long inbuf_len, /* user input buffer length */
    char *prompt,          /* prompt string */
    unsigned long prompt_len, /* prompt string length */
    unsigned long *nbytes   /* number of bytes written */
)
```

Description

This system call passes the string `prompt` to the high-level debugger (HLD) to be printed on the output screen and waits for input back from the HLD. Upon receipt of the input, it copies up to `inbuf_len` bytes into `inbuf` and writes `inbuf`'s length into `nbytes`.

Arguments

<code>inbuf</code>	Points to the buffer in which <code>db_input()</code> writes input from the HLD.
<code>inbuf_len</code>	Specifies the number of bytes of input data to be written in <code>inbuf</code> .
<code>prompt</code>	Points to the string to be passed to the HLD as a prompt.
<code>prompt_len</code>	Specifies the number of characters in <code>prompt</code> .
<code>nbytes</code>	Points to the buffer in which <code>db_input()</code> writes the actual length of <code>inbuf</code> .

Target

68K

MRI

 For 68K processors with MRI host tools, this system call is not supported.

Return Value

This call returns 0 on success and an error code on failure.

Error Codes

Hex	Mnemonic	Description
Not set at this time.	None.	Lost connection to host debugger.

See Also

db_output

db_output Outputs a string to the high-level debugger.

```
long db_output (
    char *str,           /* string */
    unsigned long length /* string length */
)
```

Description

This system call sends a character string to the high-level debugger (HLD) to be printed on its output screen.

Arguments

- str Points to the string to be output to the HLD.
- length Specifies the length of the string.

Return Value

This call returns 0 on success and an error code on failure.

Error Codes

Hex	Mnemonic	Description
Not set at this time.	None.	Lost connection to host debugger.

See Also

db_input

log_event Logs an event on ESsp's target-resident application monitor, pMONT.

```
unsigned long log_event (
    unsigned long user_event_id, /* user-defined event ID */
    unsigned long event_data    /* user-defined event data */
)
```

Description

This call logs an event in pMONT's trace buffer. The `log_event()` call takes effect when the ESsp data collection run begins.

Arguments

<code>user_event_id</code>	Specifies an ID number for the current call to <code>log_event()</code> . The maximum allowable ID number is 0xff. Providing an ID number for each call to <code>log_event()</code> helps you keep track of user-events.
<code>event_data</code>	An optional word or words you can use to store data associated with the event.

Return Value

This call always returns 0.

Error Codes

None.

Notes

Because pMONT uses pSOS+ objects for its functionality, some user-defined entries in the pSOS+ configuration table affect pMONT behavior and can even increase the likelihood of error messages. For example, an insufficient number of message buffers may result in a sudden break in the host/target connection. The paragraphs that follow explain this.

pMONT uses the system-wide buffer pool to post messages to its queues, so you need to consider this when specifying `kc_nmsgbuf` in the pSOS+ configuration

table. How much pMONT affects the specification of `kc_nmsgbuf` depends on the monitoring demands that you expect pMONT to meet.

The rate at which the system-wide buffer pool is replenished depends on the communication medium used to communicate with the host. With a network, for example, pMONT replenishes the buffer relatively quickly. You can estimate the requirements for the application by considering the following ES_p behaviors:

- A message is posted to a queue every time an object is created or deleted.
- A message is posted to a queue every time pMONT receives a request.
- A message is posted when data collection ends under any of the buffer management options.
- Under the Transmit Buffer Management option, pMONT periodically sends data to the host. Every time it does so, pMONT posts a message to a queue. The rate at which this occurs depends on the data collection configuration and the application.
- A message is posted to a queue at the end of every user-specified Perfmon update period.
- If the Stack Checking feature is on, a message is posted every time a stack warning is generated.

If the trace buffer overflows because a large number of object creates or deletes occur in a very short time, `ERR_EV_FULL` is generated, and pMONT disconnects from the ES_p tool. The way to prevent trace buffer overflow is to increase the buffer size by increasing the `kc_nlocobj` entry in the pSOS+ configuration table. However, this problem is not likely to occur in a time-critical environment.

A

Tables of System Calls

This appendix is a collection of tables with information on pSOSystem system calls, intended to help you locate a specific system call by its function rather than its name. The first table lists all pSOSystem system calls alphabetically and provides for each call a one-line description, the pSOSystem component it belongs to, and the page number where you can find more information. The remaining tables alphabetically list the system calls for each component (i.e., pSOS+, pHILE+, etc.) and provide for each call a one-line description and the page number where you can find more information.

A.1 Table of All pSOSystem Calls

TABLE A-1 All pSOSystem System Calls

Name	Component	Description	Page
abort	pREPC+	Aborts a task.	3-3
abs	pREPC+	Computes the absolute value of an integer	3-4
accept	pNA+	Accepts a connection on a socket.	4-2
access_f	pHILE+	Determines the accessibility of a file.	2-5
add_ni	pNA+	Adds a network interface.	4-4
annex_f	pHILE+	Allocates contiguous blocks to a file.	2-8
as_catch	pSOS+	Specifies an asynchronous signal routine.	1-3
asctime	pREPC+	Converts the broken-down time to a string.	3-5

TABLE A-1 All pSOSystem System Calls (Continued)

Name	Component	Description	Page
asctime_r	pREPC+	(Reentrant) Converts the broken-down time to a string.	3-6
as_return	pSOS+	Returns from an asynchronous signal routine.	1-7
as_send	pSOS+	Sends asynchronous signals to a task.	1-9
assert	pREPC+	Verifies that a program is operating correctly.	3-8
atof	pREPC+	Converts a string to a double.	3-9
atoi	pREPC+	Converts a string to an integer.	3-11
atol	pREPC+	Converts a string to a long integer.	3-13
bind	pNA+	Binds an address to a socket.	4-6
bsearch	pREPC+	Searches an array.	3-15
calloc	pREPC+	Allocates memory.	3-17
cdmount_vol	pHILE+	Mounts a CD-ROM volume	2-10
change_dir	pHILE+	Changes the current directory.	2-13
chmod_f	pHILE+	Changes the mode of a named ordinary or directory file.	2-16
chown_f	pHILE+	Changes the owner or group of a named ordinary or directory file.	2-19
clearerr	pREPC+	Clears a stream's error indicators.	3-19
close	pNA+	Closes a socket descriptor.	4-8
close_dir	pHILE+	Closes an open directory file.	2-22
close_f	pHILE+	Closes an open file connection.	2-22
connect	pNA+	Initiates a connection on a socket	4-9
create_f	pHILE+	Creates a data file.	2-25
ctime	pREPC+	Converts the calendar time to a string.	3-20

TABLE A-1 All pSOSystem System Calls (Continued)

Name	Component	Description	Page
ctime_r	pREPC+	(Reentrant) Converts the calendar time to a string.	3-22
db_input	pROBE+	Prompts and gets input from the high-level debugger.	6-3
db_output	pROBE+	Outputs a string to the high-level debugger.	6-5
de_close	pSOS+	Closes an I/O device.	1-11
de_cntrl	pSOS+	Requests a special I/O device service.	1-13
de_init	pSOS+	Initializes an I/O device and its driver.	1-15
de_open	pSOS+	Opens an I/O device.	1-17
de_read	pSOS+	Reads data from an I/O device.	1-19
de_write	pSOS+	Writes data to an I/O device.	1-21
difftime	pREPC+	Computes the difference between two calendar times.	3-24
div	pREPC+	Performs a division operation on two specified integers.	3-25
errno	pREPC+	The error number returned by the last failing system call.	3-27
errno_addr	pSOS+	Obtains the address of the calling task's internal <code>errno</code> variable.	1-23
ev_asend	pSOS+	(pSOS+m kernel only) Asynchronously sends events to a task.	1-25
ev_receive	pSOS+	Allows a task to wait for an event condition.	1-27
ev_send	pSOS+	Sends events to a task.	1-30
exit	pREPC+	Terminates a task.	3-28
fchmod_f	pHILE+	Changes the mode of an ordinary or directory file specified by its file identifier.	2-29
fchown_f	pHILE+	Changes the owner or group of a file specified by its file identifier.	2-32

A

TABLE A-1 All pSOSystem System Calls (Continued)

Name	Component	Description	Page
fclose	pREPC+	Closes a stream.	3-30
feof	pREPC+	Tests a stream's end-of-file indicator.	3-32
ferror	pREPC+	Tests a stream's error indicator.	3-33
fflush	pREPC+	Flushes the buffer associated with an open stream.	3-34
fgetc	pREPC+	Gets a character from a stream.	3-35
fgetpos	pREPC+	Gets the current file position indicator for fsetpos.	3-36
fgets	pREPC+	Gets a string from a stream.	3-37
fopen	pREPC+	Opens a file.	3-39
fprintf	pREPC+	Prints formatted output to a stream.	3-43
fputc	pREPC+	Writes a character to a stream.	3-48
fputs	pREPC+	Writes a string to a stream.	3-50
fread	pREPC+	Reads from a stream.	3-51
free	pREPC+	Deallocates memory.	3-53
freopen	pREPC+	Reopens a file.	3-54
fscanf	pREPC+	Reads formatted input from a stream.	3-56
fseek	pREPC+	Sets the file position indicator.	3-61
fsetpos	pREPC+	Sets file position by using the fgetpos result.	3-63
fstat_f	pHILE+	Obtains the status of a file specified by its file identifier.	2-35
fstat_vfs	pHILE+	Obtains statistics about a mounted volume specified by a file identifier.	2-39
ftell	pREPC+	Gets the file position indicator.	3-65
ftruncate_f	pHILE+	Changes the size of a file specified by its file identifier.	2-43

TABLE A-1 All pSOSystem System Calls (Continued)

Name	Component	Description	Page
fwrite	pREPC+	Writes to a stream.	3-67
get_fdset	pRPC+	Returns the bit mask that corresponds to readable RPC sockets.	5-5
get_fn	pHILE+	Obtains the file number of a file.	2-46
get_id	pNA+	Gets a task's user ID and group ID.	4-12
getc	pREPC+	Gets a character from a stream.	3-69
getchar	pREPC+	Gets a character from stdin.	3-70
getpeername	pNA+	Gets the address of a connected peer.	4-13
gets	pREPC+	Gets a string from stdin.	3-71
getsockname	pNA+	Gets the address that is bound to a socket.	4-15
getsockopt	pNA+	Gets options on a socket.	4-17
gmtime	pREPC+	Converts the calendar time to broken-down time.	3-72
gmtime_r	pREPC+	(Reentrant) Converts the calendar time to broken-down time.	3-73
init_vol	pHILE+	Initializes a pHILE+ formatted volume.	2-49
ioctl	pNA+	Performs control operations on a socket.	4-22
isalnum	pREPC+	Tests for an alphanumeric character.	3-75
isalpha	pREPC+	Tests for an alphabetic character.	3-76
iscntrl	pREPC+	Tests for a control character.	3-77
isdigit	pREPC+	Tests for a digit.	3-78
isgraph	pREPC+	Tests for a graphical character.	3-79
islower	pREPC+	Tests for a lowercase letter.	3-80
isprint	pREPC+	Tests for a printable character.	3-81
ispunct	pREPC+	Tests for a punctuation character.	3-82
isspace	pREPC+	Tests for a space.	3-83

A

TABLE A-1 All pSOSystem System Calls (Continued)

Name	Component	Description	Page
isupper	pREPC+	Tests for an uppercase letter.	3-84
isxdigit	pREPC+	Tests for a hexadecimal digit.	3-85
k_fatal	pSOS+	Aborts and enters fatal error handling mode.	1-40
k_terminate	pSOS+	Terminates a node other than the master node.	1-42
labs	pREPC+	Computes the absolute value of a long integer.	3-86
ldiv	pREPC+	Performs a division operation on two specified long integers.	3-87
link_f	pHILE+	Creates a hard link between two files on the same volume.	2-53
listen	pNA+	Listens for connections on a socket.	4-38
localeconv	pREPC+	Obtains the current locale settings.	3-89
localtime	pREPC+	Converts the calendar time to broken-down time.	3-92
localtime_r	pREPC+	(Reentrant) Converts the calendar time to broken-down time.	3-93
lock_f	pHILE+	Locks or unlocks part or all of an open file.	2-56
log_event	ESp	Logs an event on ESp's target-resident application monitor, pMONT.	6-6
lseek_f	pHILE+	Repositions for read or write within an open file.	2-58
lstat_f	pHILE+	Gets the status of a symbolically linked file.	2-61
m_ext2int	pSOS+	Converts an external address into an internal address.	1-44
m_int2ext	pSOS+	Converts an internal address into an external address.	1-46
make_dir	pHILE+	Creates a directory file.	2-65

TABLE A-1 All pSOSystem System Calls (Continued)

Name	Component	Description	Page
malloc	pREPC+	Allocates memory.	3-95
mblen	pREPC+	Determines the number of bytes in a multi-byte character.	3-96
mbstowcs	pREPC+	Converts a multibyte character string into a wide character string.	3-98
mbtowc	pREPC+	Converts a multibyte character into its wide character equivalent.	3-100
memchr	pREPC+	Searches memory for a character.	3-102
memcmp	pREPC+	Compares two objects in memory.	3-104
memcpy	pREPC+	Copies characters in memory.	3-106
memmove	pREPC+	Copies characters in memory.	3-108
memset	pREPC+	Initializes a memory area with a given value.	3-110
mktime	pREPC+	Converts the broken-down time into calendar time.	3-111
mount_vol	pHILE+	Mounts a pHILE+ formatted volume.	2-68
move_f	pHILE+	Moves (renames) a file.	2-70
nfsmount_vol	pHILE+	Mounts a remote file system.	2-73
open_dir	pHILE+	Opens a directory file.	2-76
open_f	pHILE+	Opens a file.	2-78
open_fn	pHILE+	Opens a file by its file identifier.	2-83
pcinit_vol	pHILE+	Initializes an MS-DOS volume.	2-85
pcmount_vol	pHILE+	Mounts an MS-DOS volume.	2-88
perror	pREPC+	Prints a diagnostic message.	3-114
pna_allocb	pNA+	Allocates a message block.	4-39
pna_esballoc	pNA+	Attaches a message block to the data buffer.	4-41
pna_freeb	pNA+	Frees a message block.	4-43

TABLE A-1 All pSOSystem System Calls (Continued)

Name	Component	Description	Page
pna_freemsg	pNA+	Frees all the message blocks associated with a message.	4-44
printf	pREPC+	Prints formatted output to <code>stdout</code> .	3-115
pt_create	pSOS+	Creates a memory partition of fixed-size buffers.	1-48
pt_delete	pSOS+	Deletes a memory partition.	1-51
pt_getbuf	pSOS+	Gets a buffer from a partition.	1-53
pt_ident	pSOS+	Obtains the identifier of the named partition.	1-55
pt_retbuf	pSOS+	Returns a buffer to the partition from which it came.	1-57
pt_sgetbuf	pSOS+	Gets a buffer from a partition.	1-59
putc	pREPC+	Writes a character to a stream.	3-117
putchar	pREPC+	Writes a character to <code>stdout</code> .	3-118
puts	pREPC+	Writes a string to a file.	3-119
q_asend	pSOS+	(pSOS+m kernel only) Asynchronously posts a message to an ordinary message queue.	1-61
q_aurgent	pSOS+	(pSOS+m kernel only) Asynchronously posts a message at the head of an ordinary message queue.	1-63
q_avsend	pSOS+	(pSOS+m kernel only) Asynchronously posts a message to a variable-length message queue.	1-65
q_avurgent	pSOS+	(pSOS+m kernel only) Asynchronously posts a message at the head of a variable-length message queue.	1-68
q_broadcast	pSOS+	Broadcasts identical messages to an ordinary message queue.	1-71
q_create	pSOS+	Creates an ordinary message queue.	1-74

TABLE A-1 All pSOSystem System Calls (Continued)

Name	Component	Description	Page
q_delete	pSOS+	Deletes an ordinary message queue.	1-77
q_ident	pSOS+	Obtains the queue ID of an ordinary message queue.	1-79
q_receive	pSOS+	Requests a message from an ordinary message queue.	1-81
q_send	pSOS+	Posts a message to an ordinary message queue.	1-84
q_urgent	pSOS+	Posts a message to the head of an ordinary message queue.	1-86
q_vbroadcast	pSOS+	Broadcasts identical variable-length messages to a variable-length message queue.	1-88
q_vcreate	pSOS+	Creates a variable-length message queue.	1-91
q_vdelete	pSOS+	Deletes a variable-length message queue.	1-94
q_vident	pSOS+	Obtains the queue ID of a variable-length message queue.	1-96
q_vreceive	pSOS+	Requests a message from a variable-length message queue.	1-98
q_vsend	pSOS+	Posts a message to a specified variable-length message queue.	1-101
q_vurgent	pSOS+	Posts a message at the head of a variable-length message queue.	1-104
qsort	pREPC+	Sorts an array in ascending order.	3-120
rand	pREPC+	Returns a pseudo-random number.	3-122
read_dir	pHILE+	Reads directory entries in a file system independent format.	2-90
read_f	pHILE+	Reads from a file.	2-93
read_link	pHILE+	Reads the value of a symbolic link.	2-96
read_vol	pHILE+	Reads directly from a pHILE+ formatted volume.	2-99

A

TABLE A-1 All pSOSystem System Calls (Continued)

Name	Component	Description	Page
realloc	pREPC+	Allocates memory.	3-123
recv	pNA+	Receives data from a socket.	4-45
recvfrom	pNA+	Receives data from a socket.	4-48
recvmsg	pNA+	Receives data from a socket.	4-51
remove	pREPC+	Removes a file.	3-125
remove_f	pHILE+	Deletes a file.	2-101
rename	pREPC+	Renames a file.	3-126
rewind	pREPC+	Resets the file position indicator.	3-128
rn_create	pSOS+	Creates a memory region.	1-107
rn_delete	pSOS+	Deletes a memory region.	1-110
rn_getseg	pSOS+	Allocates a memory segment to the calling task.	1-112
rn_ident	pSOS+	Obtains the region identifier of the named region.	1-115
rn_retseg	pSOS+	Returns a memory segment to the region from which it was allocated.	1-117
rpc_getcreateerr	pRPC+	Returns the reason for an RPC client handle creation failure.	5-6
scanf	pREPC+	Reads formatted input from stdin.	3-130
select	pNA+	Checks the status of multiple sockets.	4-54
send	pNA+	Sends data to a socket.	4-57
sendmsg	pNA+	Sends data to a socket.	4-60
sendto	pNA+	Sends data to a socket.	4-62
set_id	pNA+	Sets a task's user ID and group ID.	4-65
setbuf	pREPC+	Changes a stream's buffer.	3-132
setlocale	pREPC+	Obtains or changes the program's locale.	3-134

TABLE A-1 All pSOSystem System Calls (Continued)

Name	Component	Description	Page
setsockopt	pNA+	Sets options on a socket.	4-66
setvbuf	pREPC+	Changes a stream's buffering characteristics.	3-136
shr_socket	pNA+	Obtains a new socket descriptor for an existing socket.	4-73
shutdown	pNA+	Terminates all or part of a full-duplex connection.	4-74
sm_av	pSOS+	(pSOS+m kernel only) Asynchronously releases a semaphore token.	1-119
sm_create	pSOS+	Creates a semaphore.	1-121
sm_delete	pSOS+	Deletes a semaphore.	1-124
sm_ident	pSOS+	Obtains a semaphore identifier.	1-126
sm_p	pSOS+	Acquires a semaphore token.	1-128
sm_v	pSOS+	Releases a semaphore token.	1-131
socket	pNA+	Creates a socket.	4-76
sprintf	pREPC+	Writes formatted output to a buffer.	3-138
srand	pREPC+	Sets the seed for the random number generator (rand).	3-140
sscanf	pREPC+	Reads formatted input from a string.	3-141
stat_f	pHILE+	Gets the status of a named file.	2-104
stat_vfs	pHILE+	Gets statistics for a named volume.	2-109
strcat	pREPC+	Appends one string to another string.	3-143
strchr	pREPC+	Searches a string for a character.	3-144
strcmp	pREPC+	Compares two character strings.	3-145
strcoll	pREPC+	Compares two character strings.	3-146
strcpy	pREPC+	Copies one string to another string.	3-148

A

TABLE A-1 All pSOSystem System Calls (Continued)

Name	Component	Description	Page
strcspn	pREPC+	Calculates the length of a substring.	3-149
strerror	pREPC+	Maps an error number to an error message string.	3-150
strftime	pREPC+	Places formatted time and date information into a string.	3-151
strlen	pREPC+	Computes string length.	3-154
strncat	pREPC+	Appends characters to a string.	3-155
strncmp	pREPC+	Compares characters in two strings.	3-157
strncpy	pREPC+	Copies characters from one string to another.	3-159
strpbrk	pREPC+	Searches a string for a character in a second string.	3-160
strrchr	pREPC+	Searches a string for a character.	3-161
strspn	pREPC+	Calculates specified string length.	3-162
strstr	pREPC+	Searches a string for specified characters in another string.	3-163
strtod	pREPC+	Converts a string to a double.	3-164
strtok	pREPC+	Searches a string for tokens.	3-166
strtol	pREPC+	Converts a string to a long integer.	3-168
strtoul	pREPC+	Converts a string to an unsigned long.	3-170
strxfrm	pREPC+	Transforms a string so that it can be used by strcmp().	3-172
symlink_f	pHILE+	Creates a symbolic link to a file.	2-112
sync_vol	pHILE+	Synchronizes a volume.	2-115
t_create	pSOS+	Creates a task.	1-133
t_delete	pSOS+	Deletes a task.	1-138
t_getreg	pSOS+	Gets a task's notepad register.	1-141

TABLE A-1 All pSOSystem System Calls (Continued)

Name	Component	Description	Page
t_ident	pSOS+	Obtains the task identifier of the named task.	1-143
time	pREPC+	Obtains the current calendar time.	3-174
t_mode	pSOS+	Gets or changes the calling task's execution mode.	1-145
t_restart	pSOS+	Forces a task to start over regardless of its current state.	1-149
t_resume	pSOS+	Resumes a suspended task.	1-152
t_setpri	pSOS+	Gets and optionally changes a task's priority.	1-154
t_setreg	pSOS+	Sets a task's notepad register.	1-156
t_start	pSOS+	Starts a task.	1-158
t_suspend	pSOS+	Suspends a task until a t_resume call is made for the suspended task.	1-162
tm_cancel	pSOS+	Cancels an armed timer.	1-164
tm_evafter	pSOS+	Sends events to the calling task at periodic intervals.	1-168
tm_evevery	pSOS+	Sends events to the calling task at periodic intervals.	1-168
tm_evwhen	pSOS+	Sends events to the calling task at the specified time.	1-170
tm_get	pSOS+	Obtains the system's current version of the date and time.	1-173
tm_set	pSOS+	Sets or resets the system's version of the date and time.	1-175
tm_tick	pSOS+	Announces a clock tick to the pSOS+ kernel.	1-178
tm_wkafter	pSOS+	Blocks the calling task and wakes it after a specified interval.	1-180

A

TABLE A-1 All pSOSystem System Calls (Continued)

Name	Component	Description	Page
tm_wkwhen	pSOS+	Blocks the calling task and wakes it at a specified time.	1-182
tmpfile	pREPC+	Creates a temporary file.	3-176
tmpname	pREPC+	Generates a temporary filename.	3-177
tolower	pREPC+	Converts a character to lowercase.	3-179
toupper	pREPC+	Converts a character to uppercase.	3-180
truncate_f	pHILE+	Changes the size of a named file.	2-117
ungetc	pREPC+	Ungets a character.	3-181
unmount_vol	pHILE+	Unmounts a volume.	2-120
utime_f	pHILE+	Sets the access and modification times of a file.	2-123
verify_vol	pHILE+	Verifies a volume's control structures.	2-126
vfprintf	pREPC+	Writes formatted output to a stream.	3-183
vprintf	pREPC+	Writes formatted output to stdout.	3-185
vsprintf	pREPC+	Writes formatted output to a buffer.	3-187
wcstombs	pREPC+	Converts a wide character string into a multibyte character string.	3-189
wctomb	pREPC+	Converts a wide character into its multibyte character equivalent.	3-191
write_f	pHILE+	Writes to an open file.	2-142
write_vol	pHILE+	Writes data directly to a pHILE+ formatted volume.	2-145

A.2 pSOS+ System Calls

Table A-2 provides an alphabetical listing of all pSOS+ system calls, a summary description for each call, and a reference to more details about the call.

TABLE A-2 pSOS+ System Calls

Name	Description	Page
as_catch	Specifies an asynchronous signal routine.	1-3
as_return	Returns from an asynchronous signal routine.	1-7
as_send	Sends asynchronous signals to a task.	1-9
de_close	Closes an I/O device.	1-11
de_cntrl	Requests a special I/O device service.	1-13
de_init	Initializes an I/O device and its driver.	1-15
de_open	Opens an I/O device.	1-17
de_read	Reads data from an I/O device.	1-19
de_write	Writes data to an I/O device.	1-21
errno_addr	Obtains the address of the calling task's internal <code>errno</code> variable.	1-23
ev_asend	(pSOS+m kernel only) Asynchronously sends events to a task.	1-25
ev_receive	Allows a task to wait for an event condition.	1-27
ev_send	Sends events to a task.	1-30
i_enter	Enters into an interrupt service routine.	1-33
i_return	Provides an exit from an interrupt service routine.	1-35
k_fatal	Aborts and enters fatal error handling mode.	1-40
k_terminate	Terminates a node other than the master node.	1-42
m_ext2int	Converts an external address into an internal address.	1-44
m_int2ext	Converts an internal address into an external address.	1-46
pt_create	Creates a memory partition of fixed-size buffers.	1-48
pt_delete	Deletes a memory partition.	1-51

TABLE A-2 pSOS+ System Calls (Continued)

Name	Description	Page
pt_getbuf	Gets a buffer from a partition.	1-53
pt_ident	Obtains the identifier of the named partition.	1-55
pt_retbuf	Returns a buffer to the partition from which it came.	1-57
pt_sgetbuf	Gets a buffer from a partition.	1-59
q_asend	(pSOS+m kernel only) Asynchronously posts a message to an ordinary message queue.	1-61
q_urgent	(pSOS+m kernel only) Asynchronously posts a message at the head of an ordinary message queue.	1-63
q_avsend	(pSOS+m kernel only) Asynchronously posts a message to a variable-length message queue.	1-65
q_avurgent	(pSOS+m kernel only) Asynchronously posts a message at the head of a variable-length message queue.	1-68
q_broadcast	Broadcasts identical messages to an ordinary message queue.	1-71
q_create	Creates an ordinary message queue.	1-74
q_delete	Deletes an ordinary message queue.	1-77
q_ident	Obtains the queue ID of an ordinary message queue.	1-79
q_receive	Requests a message from an ordinary message queue.	1-81
q_send	Posts a message to an ordinary message queue.	1-84
q_urgent	Posts a message to the head of an ordinary message queue.	1-86
q_vbroadcast	Broadcasts identical variable-length messages to a variable-length message queue.	1-88
q_vcreate	Creates a variable-length message queue.	1-91
q_vdelete	Deletes a variable-length message queue.	1-94
q_vident	Obtains the queue ID of a variable-length message queue.	1-96
q_vreceive	Requests a message from a variable-length message queue.	1-98
q_vsend	Posts a message to a specified variable-length message queue.	1-101

TABLE A-2 pSOS+ System Calls (Continued)

Name	Description	Page
q_vurgent	Posts a message at the head of a variable-length message queue.	1-104
rn_create	Creates a memory region.	1-107
rn_delete	Deletes a memory region.	1-110
rn_getseg	Allocates a memory segment to the calling task.	1-112
rn_ident	Obtains the region identifier of the named region.	1-115
rn_retseg	Returns a memory segment to the region from which it was allocated.	1-117
sm_av	(pSOS+m kernel only) Asynchronously releases a semaphore token.	1-119
sm_create	Creates a semaphore.	1-121
sm_delete	Deletes a semaphore.	1-124
sm_ident	Obtains a semaphore identifier.	1-126
sm_p	Acquires a semaphore token.	1-128
sm_v	Releases a semaphore token.	1-131
t_create	Creates a task.	1-133
t_delete	Deletes a task.	1-138
t_getreg	Gets a task's notepad register.	1-141
t_ident	Obtains the task identifier of the named task.	1-143
t_mode	Gets or changes the calling task's execution mode.	1-145
t_restart	Forces a task to start over regardless of its current state.	1-149
t_resume	Resumes a suspended task.	1-152
t_setpri	Gets and optionally changes a task's priority.	1-154
t_setreg	Sets a task's notepad register.	1-156
t_start	Starts a task.	1-158

A

TABLE A-2 pSOS+ System Calls (Continued)

Name	Description	Page
t_suspend	Suspends a task until a t_resume call is made for the suspended task.	1-162
tm_cancel	Cancels an armed timer.	1-164
tm_evafter	Sends events to the calling task after a specified interval.	1-166
tm_evevery	Sends events to the calling task at periodic intervals.	1-168
tm_evwhen	Sends events to the calling task at the specified time.	1-170
tm_get	Obtains the system's current version of the date and time.	1-173
tm_set	Sets or resets the system's version of the date and time.	1-175
tm_tick	Announces a clock tick to the pSOS+ kernel.	1-178
tm_wkafter	Blocks the calling task and wakes it after a specified interval.	1-180
tm_wkwhen	Blocks the calling task and wakes it at a specified time.	1-182

A.3 pHILE+ System Calls

Table A-3 provides an alphabetical listing of all pHILE+ system calls, a summary description for each call, and a reference to more details about the call.

TABLE A-3 pHILE+ System Calls

Name	Description	Page
access_f	Determines the accessibility of a file.	2-5
annex_f	Allocates contiguous blocks to a file.	2-8
cdmount_vol	Mounts a CD-ROM volume	2-10
change_dir	Changes the current directory.	2-13
chmod_f	Changes the mode of a named file.	2-16
chown_f	Changes the owner or group of a named file.	2-19
close_dir	Closes an open directory file.	2-22

TABLE A-3 pHILE+ System Calls (Continued)

Name	Description	Page
close_f	Closes an open file connection.	2-22
create_f	Creates a data file.	2-25
fchmod_f	Changes the mode of a file specified by its file identifier.	2-29
fchown_f	Changes the owner or group of a file specified by its file identifier.	2-32
fstat_f	Obtains the status of a file specified by its file identifier.	2-35
fstat_vfs	Obtains statistics about a mounted volume specified by a file identifier.	2-39
ftruncate_f	Changes the size of a file specified by its file identifier.	2-43
get_fn	Obtains the file number of a file.	2-46
init_vol	Initializes a pHILE+ formatted volume.	2-49
link_f	Creates a hard link between two files on the same volume.	2-53
lock_f	Locks or unlocks part or all of an open file.	2-56
lseek_f	Repositions for read or write within an open file.	2-58
lstat_f	Gets the status of a symbolically linked file.	2-61
make_dir	Creates a directory file.	2-65
mount_vol	Mounts a pHILE+ formatted volume.	2-68
move_f	Moves (renames) a file.	2-70
nfsmount_vol	Mounts a remote file system.	2-73
open_dir	Opens a directory file.	2-76
open_f	Opens a file.	2-78
open_fn	Opens a file by its file identifier.	2-83
pcinit_vol	Initializes an MS-DOS volume.	2-85
pcmount_vol	Mounts an MS-DOS volume.	2-88
read_dir	Reads directory entries in a file system independent format.	2-90

A

TABLE A-3 pHILE+ System Calls (Continued)

Name	Description	Page
read_f	Reads from a file.	2-93
read_link	Reads the value of a symbolic link.	2-96
read_vol	Reads directly from a pHILE+ formatted volume.	2-99
remove_f	Deletes a file.	2-101
stat_f	Gets the status of a named file.	2-104
stat_vfs	Gets statistics for a named volume.	2-109
symlink_f	Creates a symbolic link to a file.	2-112
sync_vol	Synchronizes a volume.	2-115
truncate_f	Changes the size of a named file.	2-117
unmount_vol	Unmounts a volume.	2-120
utime_f	Sets the access and modification times of a file.	2-123
verify_vol	Verifies a volume's control structures.	2-126
write_f	Writes to an open file.	2-142
write_vol	Writes data directly to a pHILE+ formatted volume.	2-145

A.4 pREPC+ System Calls

Table A-4 provides an alphabetical listing of all pREPC+ system calls, a summary description for each call, and a reference to more details about the call.

TABLE A-4 pREPC+ System Calls

Name	Description	Page
abort	Aborts a task.	3-3
abs	Computes the absolute value of an integer.	3-4
asctime	Converts the broken-down time to a string.	3-5

TABLE A-4 pREPC+ System Calls (Continued)

Name	Description	Page
asctime_r	(Reentrant) Converts the broken-down time to a string.	3-6
assert	Verifies that a program is operating correctly.	3-8
atof	Converts a string to a double.	3-9
atoi	Converts a string to an integer.	3-11
atol	Converts a string to a long integer.	3-13
bsearch	Searches an array.	3-15
calloc	Allocates memory.	3-17
clearerr	Clears a stream's error indicators.	3-19
ctime	Converts the calendar time to a string.	3-20
ctime_r	(Reentrant) Converts the calendar time to a string.	3-22
difftime	Computes the difference between two calendar times.	3-24
div	Performs a division operation on two specified integers.	3-25
errno	The error number returned by the last failing system call.	3-27
exit	Terminates a task.	3-28
fclose	Closes a stream.	3-30
feof	Tests a stream's end-of-file indicator.	3-32
ferror	Tests a stream's error indicator.	3-33
fflush	Flushes the buffer associated with an open stream.	3-34
fgetc	Gets a character from a stream.	3-35
fgetpos	Gets the current file position indicator for fsetpos.	3-36
fgets	Gets a string from a stream.	3-37
fopen	Opens a file.	3-39
fprintf	Prints formatted output to a stream.	3-43
fputc	Writes a character to a stream.	3-48

A

TABLE A-4 pREPC+ System Calls (Continued)

Name	Description	Page
fputs	Writes a string to a stream.	3-50
fread	Reads from a stream.	3-51
free	Deallocates memory.	3-53
freopen	Reopens a file.	3-54
fscanf	Reads formatted input from a stream.	3-56
fseek	Sets the file position indicator.	3-61
fsetpos	Sets file position by using the fgetpos result.	3-63
ftell	Gets the file position indicator.	3-65
fwrite	Writes to a stream.	3-67
getc	Gets a character from a stream.	3-69
getchar	Gets a character from stdin.	3-70
gets	Gets a string from stdin.	3-71
gmtime	Converts the calendar time to broken-down time.	3-72
gmtime_r	(Reentrant) Converts the calendar time to broken-down time.	3-73
isalnum	Tests for an alphanumeric character.	3-75
isalpha	Tests for an alphabetic character.	3-76
iscntrl	Tests for a control character.	3-77
isdigit	Tests for a digit.	3-78
isgraph	Tests for a graphical character.	3-79
islower	Tests for a lowercase letter.	3-80
isprint	Tests for a printable character.	3-81
ispunct	Tests for a punctuation character.	3-82
isspace	Tests for a space.	3-83
isupper	Tests for an uppercase letter.	3-84

TABLE A-4 pREPC+ System Calls (Continued)

Name	Description	Page
isxdigit	Tests for a hexadecimal digit.	3-85
labs	Computes the absolute value of a long integer.	3-86
ldiv	Performs a division operation on two specified long integers.	3-87
localeconv	Obtains the current locale settings.	3-89
localtime	Converts the calendar time to broken-down time.	3-92
localtime_r	(Reentrant) Converts the calendar time to broken-down time.	3-93
malloc	Allocates memory.	3-95
mblen	Determines the number of bytes in a multibyte character.	3-96
mbstowcs	Converts a multibyte character string into a wide character string.	3-98
mbtowc	Converts a multibyte character into its wide character equivalent.	3-100
memchr	Searches memory for a character.	3-102
memcmp	Compares two objects in memory.	3-104
memcpy	Copies characters in memory.	3-106
memmove	Copies characters in memory.	3-108
memset	Initializes a memory area with a given value.	3-110
mktime	Converts the broken-down time into calendar time.	3-111
perror	Prints a diagnostic message.	3-114
printf	Prints formatted output to stdout.	3-115
putc	Writes a character to a stream.	3-117
putchar	Writes a character to stdout.	3-118
puts	Writes a string to a stream.	3-119
qsort	Sorts an array in ascending order.	3-120
rand	Returns a pseudo-random number.	3-122

A

TABLE A-4 pREPC+ System Calls (Continued)

Name	Description	Page
realloc	Allocates memory.	3-123
remove	Removes a file.	3-125
rename	Renames a file.	3-126
rewind	Resets the file position indicator.	3-128
scanf	Reads formatted input from stdin.	3-130
setbuf	Changes a stream's buffer.	3-132
setlocale	Obtains or changes the program's locale.	3-134
setvbuf	Changes a stream's buffering characteristics.	3-136
sprintf	Writes formatted output to a buffer.	3-138
srand	Sets the seed for the random number generator (rand.)	3-140
sscanf	Reads formatted input from a string.	3-141
strcat	Appends one string to another string.	3-143
strchr	Searches a string for a character.	3-144
strcmp	Compares two character strings.	3-145
strcoll	Compares two character strings.	3-146
strcpy	Copies one string to another string.	3-148
strcspn	Calculates the length of a substring.	3-149
strerror	Maps an error number to an error message string.	3-150
strftime	Places formatted time and date information into a string.	3-151
strlen	Computes string length.	3-154
strncat	Appends characters to a string.	3-155
strncmp	Compares characters in two strings.	3-157
strncpy	Copies characters from one string to another.	3-159
strpbrk	Searches a string for a character in a second string.	3-160

TABLE A-4 pREPC+ System Calls (Continued)

Name	Description	Page
<code>strrchr</code>	Searches a string for a character.	3-161
<code>strspn</code>	Calculates specified string length.	3-162
<code>strstr</code>	Searches a string for specified characters in another string.	3-163
<code>strtod</code>	Converts a string to a double.	3-164
<code>strtok</code>	Searches a string for tokens.	3-166
<code>strtol</code>	Converts a string to a long integer.	3-168
<code>strtoul</code>	Converts a string to an unsigned long.	3-170
<code>strxfrm</code>	Transforms a string so that it can be used by <code>strcmp()</code> .	3-172
<code>time</code>	Obtains the current calendar time.	3-174
<code>tmpfile</code>	Creates a temporary file.	3-176
<code>tmpname</code>	Generates a temporary file name.	3-177
<code>tolower</code>	Converts a character to lowercase.	3-179
<code>toupper</code>	Converts a character to uppercase.	3-180
<code>ungetc</code>	Ungets a character.	3-181
<code>vfprintf</code>	Writes formatted output to a stream.	3-183
<code>vprintf</code>	Writes formatted output to <code>stdout</code> .	3-185
<code>vsprintf</code>	Writes formatted output to a buffer.	3-187
<code>wcstombs</code>	Converts a wide character string into a multibyte character string.	3-189
<code>wctomb</code>	Converts a wide character into its multibyte character equivalent.	3-191

A

A.5 pNA+ System Calls

Table A-5 provides an alphabetical listing of all pNA+ system calls, a summary description for each call, and a reference to more details about the call.

TABLE A-5 pNA+ System Calls

Name	Description	Page
accept	Accepts a connection on a socket.	4-2
add_ni	Adds a network interface.	4-4
bind	Binds an address to a socket.	4-6
close	Closes a socket descriptor.	4-8
connect	Initiates a connection on a socket	4-9
get_id	Gets a task's user ID and group ID.	4-12
getpeername	Gets the address of a connected peer.	4-13
getsockname	Gets the address that is bound to a socket.	4-15
getsockopt	Gets options on a socket.	4-17
ioctl	Performs control operations on a socket.	4-22
listen	Listens for connections on a socket.	4-38
pna_allocb	Allocates a message block.	4-39
pna_esballoc	Attaches a message block to the data buffer.	4-41
pna_freeb	Frees a message block.	4-43
pna_freemsg	Frees all the message blocks associated with a message.	4-44
recv	Receives data from a socket.	4-45
recvfrom	Receives data from a socket.	4-48
recvmsg	Receives data from a socket.	4-51
select	Checks the status of multiple sockets.	4-54
send	Sends data to a socket.	4-57
sendmsg	Sends data to a socket.	4-60

TABLE A-5 pNA+ System Calls (Continued)

Name	Description	Page
<code>sendto</code>	Sends data to a socket.	4-62
<code>set_id</code>	Sets a task's user ID and group ID.	4-65
<code>setsockopt</code>	Sets options on a socket.	4-66
<code>shr_socket</code>	Obtains a new socket descriptor for an existing socket.	4-73
<code>shutdown</code>	Terminates all or part of a full-duplex connection.	4-74
<code>socket</code>	Creates a socket.	4-76

A.6 pRPC+ System Calls

Table A-6 provides an alphabetical listing of the pRPC+ system calls described in this manual, a summary description for each call, and a reference to more details about the call.

TABLE A-6 pRPC+ System Calls

Name	Description	Page
<code>get_fdset</code>	Returns the bit mask that corresponds to readable RPC sockets.	5-5
<code>rpc_getcreateerr</code>	Returns the reason for an RPC client handle creation failure.	5-6

A.7 pROBE+ and ESsp System Calls

Table A-7 provides an alphabetical listing of the system calls supported by pROBE+ and ESsp. Each call listing includes the call name, the tool name, a summary description, and a reference to more details about the call.

TABLE A-7 pROBE and ESsp System Calls

Name	Tool	Description	Page
db_input	pROBE+	Prompts and gets input from the high-level debugger.	6-3
db_output	pROBE+	Outputs a string to the high-level debugger.	6-5
log_event	ESsp	Logs an event on ESsp's target-resident application monitor, pMONT.	6-6

B

Error Codes

This appendix is a collection of tables of pSOSystem error codes, intended to help you identify which system call returned a specific error code. Each table lists the codes belonging to a single pSOSystem component (i.e., pSOS+, pHILE+, etc.) The table entry for each code includes a hexadecimal number, a brief description (including the error mnemonic), and a list of the system calls that can return the error.

pSOSystem components return error codes in two ways:

- pSOS+ and pHILE+ return error codes as function return values.
- pREPC+, pNA+, and pRPC+ load the error code into an internal variable that can be read through the macro `errno()`. If the return value of a pREPC+, pNA+, or pRPC+ system call indicates an error, your application should examine the `errno` variable to determine the cause of the error. See the description of `errno()` on page 3-27 for more information.

Table B-1 lists the error code ranges of pSOSystem components, libraries, and drivers. Error code values are in hexadecimal notation, with a space inserted every byte for readability.

TABLE B-1 Error Code Origins

Error Code Range		Origin	Defined in	Refer to page
From	To			
00 00 00 01	00 00 0F FF	pSOS+, pSOS+m	<psos.h>	B-4
00 00 10 00	00 00 1F FF	(reserved)		
00 00 20 00	00 00 2F FF	pHILE+	<phile.h>	B-15

TABLE B-1 Error Code Origins (Continued)

Error Code Range		Origin	Defined in	Refer to page
From	To			
00 00 30 00	00 00 3F FF	pREPC+	<errno.h>	B-38
00 00 40 00	00 00 4F FF	(reserved)		
00 00 50 00	00 00 5F FF	pNA+, pRPC+	<pna.h>	B-39
00 00 60 00	00 00 6F FF	(reserved)		
00 01 00 00	00 FF FF FF	(reserved)		
01 10 00 00	01 1F FF FF	Networking libraries		
01 20 00 00	01 20 00 FF	MMUlib	<mmulib.h>	
01 20 01 00	01 20 01 FF	Loader		
01 20 02 00	00 FF FF FF	(reserved for pSOSystem libraries)		
10 00 00 00	10 00 00 FF	NI_SMEM driver	<drv_intf.h>	B-45
10 00 01 00	10 00 01 FF	KI_SMEM driver	<drv_intf.h>	B-45
10 00 02 00	10 00 FF FF	(reserved for drivers)		
10 01 00 00	10 01 FF FF	serial driver	<drv_intf.h>	B-46
10 02 00 00	10 02 FF FF	tick timer driver	<drv_intf.h>	B-47
10 03 00 00	10 03 FF FF	(reserved for drivers)		
10 04 00 00	10 04 FF FF	RAM disk driver	<drv_intf.h>	B-48
10 05 00 00	10 05 FF FF	(reserved for drivers)		
10 06 00 00	10 06 FF FF	TFTP driver	<drv_intf.h>	B-48
10 07 00 00	10 07 FF FF	SLIP driver		
10 08 00 00	10 08 FF FF	(reserved for drivers)		
10 09 00 00	10 09 FF FF	IDE driver		B-49
10 0A 00 00	10 0A FF FF	FLP driver		B-49

TABLE B-1 Error Code Origins (Continued)

Error Code Range		Origin	Defined in	Refer to page
From	To			
10 0B 00 00	10 4F FF FF	(reserved for drivers)		
10 50 00 00	10 5F FF FF	SCSI driver	<drv_intf.h>	B-50
10 60 00 00	1F FF FF FF	(reserved for drivers)		
20 00 00 00	FF FF FF FF	(reserved for application use)		

B.1 pSOS+ Error Codes

All pSOS+ error codes are returned as function return values (rather than an `errno` variable); they have a value between 0 and 0xffff.

Table B-2 lists all error codes returned by the pSOS+ component. Each listing includes the error code's hexadecimal number, its mnemonic and description, and the pSOS+ system calls that can return it. The error code mnemonics are also defined in `<psos.h>`.

The term *object* represents the applicable service group type (task, partition, queue, semaphore, and so on).

TABLE B-2 pSOS+ Error Codes

Hex	Mnemonic and Description	System Call(s)
0x01	ERR_TIMEOUT: Timed out; returned only if a timeout was requested.	ev_receive, q_receive, q_vreceive, rn_getseg. sm_p
0x03	ERR_SSFN: Illegal system service function number.	ev_asend, q_asend, q_aurgent, q_avsend, q_avurgent, sm_av
0x04	ERR_NODENO: Node specifier out of range.	k_terminate, pt_ident, q_ident, q_vident, sm_ident, t_ident

TABLE B-2 pSOS+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x05	ERR_OBJDEL: <i>object</i> has been deleted.	as_send, ev_asend, ev_send, pt_delete, pt_getbuf, pt_retbuf, pt_sgetbuf, q_asend, q_urgent, q_avsend, q_avurgent, q_broadcast, q_delete, q_receive, q_send, q_urgent, q_vbroadcast, q_vdelete, q_vreceive, q_vsend, q_vurgent, rn_delete, rn_getseg, rn_retseg, sm_av, sm_delete, sm_p, sm_v, t_delete, t_getreg, t_restart, t_resume, t_setpri, t_setreg, t_start, t_suspend,

B

TABLE B-2 pSOS+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x06	ERR_OBJID: <i>object_id</i> is incorrect; failed validity check.	as_send, ev_asend, ev_send, pt_delete, pt_getbuf, pt_retbuf, pt_sgetbuf, q_asend, q_aurgent, q_avsend, q_avurgent, q_broadcast, q_delete, q_receive, q_send, q_urgent, q_vbroadcast, q_vdelete, q_vreceive, q_vsend, q_vurgent, rn_delete, rn_getseg, rn_retseg, sm_av, sm_delete, sm_p, sm_v, t_delete, t_getreg, t_restart, t_resume, t_setpri, t_setreg, t_start, t_suspend

TABLE B-2 pSOS+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x07	ERR_OBJTYPE: <i>object</i> type doesn't match <i>object</i> ID; failed validity check.	as_send, ev_asend, ev_send, pt_delete, pt_getbuf, pt_retbuf, pt_sgetbuf, q_asend, q_urgent, q_avsend, q_avurgent, q_broadcast, q_delete, q_receive, q_send, q_urgent, q_vbroadcast, q_vdelete, q_vreceive, q_vsend, q_vurgent, rn_delete, rn_getseg, rn_retseg, sm_av, sm_delete, sm_p, sm_v, t_delete, t_getreg, t_restart, t_resume, t_setpri, t_setreg, t_start, t_suspend
0x08	ERR_OBJTFULL: Node's <i>object</i> table full.	pt_create, q_create, q_vcreate, rn_create, sm_create, t_create,
0x09	ERR_OBJJNF: Named <i>object</i> not found.	pt_ident, q_ident, q_vident, rn_ident, sm_ident, t_ident
0x0D	ERR_RSTFS: Informative; files may be corrupted on restart.	t_restart
0x0E	ERR_NOTCB: Exceeds node's maximum number of tasks.	t_create
0x0F	ERR_NOSTK: Insufficient space in Region 0 to create stack or task started in user mode with no user stack allocated.	t_create, t_start

TABLE B-2 pSOS+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x10	ERR_TINYSTK: Stack too small.	t_create
0x11	ERR_PRIOR: Priority out of range.	t_create
0x12	ERR_ACTIVE: Task already started.	t_start
0x13	ERR_NACTIVE: Cannot restart; this task never was started.	t_restart
0x14	ERR_SUSP: Task already suspended.	t_suspend
0x15	ERR_NOTSUSP: The task was not suspended.	t_resume
0x16	ERR_SETPRI: Cannot change priority; new priority out of range.	t_setpri
0x17	ERR_REGNUM: Register number out of range.	t_getreg, t_setreg
0x18	ERR_DELFs: pHILE+ resources in use.	t_delete
0x19	ERR_DELLC: pREPC+ resources in use.	t_delete
0x1A	ERR_DELNS: pNA+ resources in use.	t_delete
0x1B	ERR_RNADDR: Starting address not on long word boundary.	rn_create
0x1C	ERR_UNITSIZE: Illegal <i>unit_size</i> — unit size not power of 2 or less than 16 bytes.	rn_create
0x1D	ERR_TINYUNIT: <i>length</i> too large (for given <i>unit_size</i> .)	rn_create
0x1E	ERR_TINYRN: Cannot create; region length too small to hold RNCB.	rn_create
0x1F	ERR_SEGINUSE: Cannot delete; one or more segments still in use.	rn_delete
0x20	ERR_ZERO: Cannot getseg; request size of zero is illegal.	rn_getseg
0x21	ERR_TOOBIG: Cannot getseg; request size is too big for region.	rn_getseg

TABLE B-2 pSOS+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x22	ERR_NOSEG: No free segment; only if <i>RN_NOWAIT</i> attribute used.	rn_getseg
0x23	ERR_NOTINRN: Segment does not belong to this region.	rn_retseg
0x24	ERR_SEGADDR: Incorrect segment starting address.	rn_retseg
0x25	ERR_SEGFREE: Segment is already unallocated.	rn_retseg
0x26	ERR_RNKILLD: Cannot getseg; region deleted while waiting.	rn_getseg
0x27	ERR_TATRNDL: Informative only; there were tasks waiting.	rn_delete
0x28	ERR_PTADDR: Starting address not on long word boundary.	pt_create
0x29	ERR_BUFSIZE: Buffer size not power of 2, or less than 4 bytes.	pt_create
0x2A	ERR_TINYPT: Length too small to hold PTCB.	pt_create
0x2B	ERR_BUFINUSE: Cannot delete; one or more buffers still in use.	pt_delete
0x2C	ERR_NOBUF: Cannot allocate; partition out of free buffers.	pt_getbuf, pt_sgetbuf
0x2D	ERR_BUFADDR: Incorrect buffer starting address.	pt_retbuf
0x2F	ERR_BUFFREE: Buffer is already unallocated.	pt_retbuf
0x30	ERR_KISIZE: Global queue maxlen too large for KI.	q_vbroadcast, q_vcreate, q_vreceive, q_vsend, q_vurgent
0x31	ERR_MSGSIZ: Message too large.	q_vsend, q_vurgent, q_vbroadcast
0x32	ERR_BUFSIZ: Buffer too small.	q_vreceive, pt_create

B

TABLE B-2 pSOS+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x33	ERR_NOQCB: Can't allocate QCB: exceeds node's active queue maximum.	q_create, q_vcreate
0x34	ERR_NOMGB: Cannot allocate private buffers; too few available.	q_asend, q_aurgent, q_create, q_send, q_urgent, q_vcreate, errno_addr
0x35	ERR_QFULL: Message queue at length limit.	q_asend, q_aurgent, q_avsend, q_avurgent, q_send, q_urgent, q_vsend, q_vurgent
0x36	ERR_QKILLD: Queue deleted while task waiting.	q_receive, q_vreceive
0x37	ERR_NOMSG: Queue empty: this error returns only if Q_NOWAIT selected.	q_receive, q_vreceive
0x38	ERR_TATQDEL: Informative only: tasks were waiting at the queue.	q_delete, q_vdelete
0x39	ERR_MATQDEL: Information only: messages were pending in the queue.	q_delete, q_vdelete
0x3A	ERR_VARQ: Queue is variable length.	q_asend, q_aurgent, q_broadcast, q_delete, q_receive, q_send, q_urgent
0x3B	ERR_NOTVARQ: Queue is not variable length.	q_vbroadcast, q_vdelete, q_vreceive, q_vsend, q_vurgent, q_avsend, q_avurgent
0x3C	ERR_NOEVS: Selected events not pending; this error code is returned only if the EV_NOWAIT attribute was selected.	ev_receive
0x3E	ERR_NOTINASR: Illegal, not called from an ASR.	as_return
0x3F	ERR_NOASR: Task has no valid ASR.	as_send

TABLE B-2 pSOS+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x41	ERR_NOSCB: Exceeds node's maximum number of semaphores.	sm_create
0x42	ERR_NOSEM: No semaphore: this error code returns only if SM_NOWAIT was selected.	sm_p
0x43	ERR_SKILLD: Semaphore deleted while task waiting.	sm_p
0x44	ERR_TATSDEL: Informative only; there were tasks waiting.	sm_delete
0x47	ERR_NOTIME: System time and date not yet set.	tm_evafter, tm_evevery, tm_evwhen, tm_get, tm_wkwhen
0x48	ERR_ILLDATE: <i>date</i> input out of range.	tm_evwhen, tm_set, tm_wkwhen
0x49	ERR_ILLTIME: <i>time</i> input out of range.	tm_evwhen, tm_set, tm_wkwhen
0x4A	ERR_ILLTICKS: <i>ticks</i> input out of range.	tm_evwhen, tm_set, tm_wkwhen
0x4B	ERR_NOTIMERS: Exceeds maximum number of configured timers.	tm_evafter, tm_evevery, tm_evwhen
0x4C	ERR_BADTMID: <i>tmid</i> invalid.	tm_cancel
0x4D	ERR_TMNOTSET: Timer not armed or already expired.	tm_cancel
0x4E	ERR_TOOLATE: Too late; <i>date</i> and <i>time</i> input already in the past.	tm_evwhen, tm_wkwhen
0x53	ERR_ILLRSC: <i>object</i> not created from this node.	pt_delete, q_delete, q_vdelete, sm_delete, t_delete, t_restart, t_start
0x54	ERR_NOAGNT: Cannot wait; the remote node is out of Agents.	q_receive, q_vreceive

TABLE B-2 pSOS+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x65	ERR_STALEID: <i>object</i> does not exist any more.	ev_send, pt_getbuf, pt_retbuf, pt_sgetbuf, q_asend, q_aurgent, q_avsend, q_avurgent, q_broadcast, q_receive, q_send, q_urgent, q_vbroadcast, q_vreceive, q_vsend, q_vurgent, sm_p, sm_v, t_getreg, t_resume, t_setpri, t_setreg,
0x66	ERR_NDKLD: Remote node is no longer in service.	q_receive, q_vreceive, sm_p
0x67	ERR_MASTER: Cannot terminate master node.	k_terminate
0x101	ERR_IODN: Illegal device (major) number.	de_close, de_cntrl, de_init, de_open, de_read, de_write
0x102	ERR_NODR: No driver provided.	de_close, de_cntrl, de_init, de_open, de_read, de_write
0x103	ERR_IOOP: Illegal I/O function number.	de_close, de_cntrl, de_init, de_open, de_read, de_write
0xF00	FAT_ALIGN: Region 0 must be aligned on a long word boundary.	This error originates in pSOS+ initialization.
0xF01	FAT_OVSDA: Region 0 overflow while making system data area.	This error originates in pSOS+ initialization.
0xF02	FAT_OVOBJT: Region 0 overflow while making object table.	This error originates in pSOS+ initialization.
0xF03	FAT_OVDDAT: Region 0 overflow while making device data area table.	This error originates in pSOS+ initialization.

TABLE B-2 pSOS+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0xF04	FAT_OVTCB: Region 0 overflow while making task structures.	This error originates in pSOS+ initialization.
0xF05	FAT_OVQCB: Region 0 overflow while making queue structures.	This error originates in pSOS+ initialization.
0xF06	FAT_OVSMCB: Region 0 overflow while making semaphore structures.	This error originates in pSOS+ initialization.
0xF07	FAT_OVTM: Region 0 overflow while making timer structures.	This error originates in pSOS+ initialization.
0xF08	FAT_OVPT: Region 0 overflow while making partition structures.	This error originates in pSOS+ initialization.
0xF09	FAT_OVRSC: Region 0 overflow while making RSC structures.	This error originates in pSOS+ initialization.
0xF0A	FAT_OVRN: Region 0 overflow while making region structures.	This error originates in pSOS+ initialization.
0xF0C	FAT_ROOT: Cannot create ROOT task.	This error originates in pSOS+ initialization.
0xF0D	FAT_IDLE: Cannot create IDLE task.	This error originates in pSOS+ initialization.
0xF0E	FAT_CHKSUM: Checksum error.	This error originates in pSOS+ initialization.
0xF0F	FAT_INVCPU: Wrong processor type.	This error originates in pSOS+ initialization.
0xF12	FAT_ILLPKT: Illegal packet type in the received packet.	This error originates in pSOS+ initialization.
0xF13	FAT_MIVERIF: Multiprocessor configuration mismatch at system verify.	This error originates in pSOS+ initialization.
0xF15	FAT_NODENUM: Illegal value for mc_nodenum.	This error originates in pSOS+ initialization.
0xF16	FAT_NNODES: Illegal value for mc_nnodes.	This error originates in pSOS+ initialization.

B

TABLE B-2 pSOS+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0xF17	FAT_OVMP: Region 0 overflow while making multiprocessor structures.	This error originates in pSOS+ initialization.
0xF18	FAT_KIMAXBUF: mc_kimaxbuf too small for mc_nnodes.	This error originates in pSOS+ initialization.
0xF19	FAT_ASYNCERR: Asynchronous RSC failure.	This error originates in pSOS+ initialization.
0xF1B	FAT_DEVINIT: Error during auto-initialization of a device.	This error originates in pSOS+ initialization.
0xF20	FAT_JN2SOON: Join request denied — Node already in system.	This error originates in pSOS+ initialization.
0xF21	FAT_MAXSEQ: Join request denied — Sequence number at limit.	This error originates in pSOS+ initialization.
0xF22	FAT_JRQATSLV: Join request sent to a slave node instead of the Master node.	This error originates in pSOS+ initialization.

B.2 pHILE+ Error Codes

pHILE+ error codes are returned as function return values (rather than an `errno` variable). Table B-3 lists all error codes returned by the pHILE+ component. Each listing includes the error code's hexadecimal number, it's mnemonic and description, and the pHILE+ system calls that can return it. The error code mnemonics are also defined in `<phile.h>`.

An asterisk next to an error code's description indicates that it can represent an NFS or RPC error. Sections B.2.2 and B.2.3 beginning on page B-35 provide tables of the NFS and RPC error codes that are mapped to pHILE+ error codes.

TABLE B-3 pHILE+ Error Codes

Hex	Mnemonic and Description	System Call(s)
0x2001	E_FUNC: Invalid function number. The function number passed to pHILE+ in register D0.L does not contain a code corresponding to a valid pHILE+ system call.	access_f, chmod_f, chown_f, fchmod_f, fchown_f, ftruncate_f, link_f, lstat_f, make_dir, read_link, symlink_f, truncate_f, utime_f
0x2002	E_FAIL: pHILE+ failure. An internal error has been detected by the pHILE+ file system manager. Report this error condition to Integrated Systems.	Should never happen.
0x2003	E_BADVOL: Inconsistent data on volume; volume corrupted. The data structures on the volume are inconsistent with each other. This is most likely the result of a crash while the pHILE+ file system manager was writing to the volume. On MS-DOS volumes, it can also indicate that an incorrect partition number has been specified.	change_dir, close_dir, close_f, create_f, ftruncate_f, get_fn, init_vol, make_dir, move_f, open_dir, open_f, open_fn, pcinit_vol, pcmount_vol, read_dir, read_f, remove_f, stat_f, stat_vfs, sync_vol, truncate_f, unmount_vol, write_f

B

TABLE B-3 pHILE+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x2005	<p>E_VINITPAR: Illegal parameters to <code>init_vol()</code>. The parameters specified to an <code>init_vol()</code> call are not consistent. One of the following problems has been detected by the pHILE+ file system manager.</p> <ul style="list-style-type: none"> ■ The specified starting location of the bitmap causes the FLIST to either extend beyond the end of the volume or the end of the control block region (if the volume has been partitioned into control and data block regions.) ■ The specified starting block for the data block region (SODATA) is not on a modulo 8 boundary, or it is beyond the end of the volume. ■ The bitmap begins in blocks 0-3. 	<code>init_vol</code>
0x2006	<p>E_MNTFULL: Attempt to mount too many volumes. Attempt to mount more volumes than specified by the pHILE+ Configuration Table parameter <code>fc_nmount</code>.</p>	<code>cdmount_vol</code> , <code>mount_vol</code> , <code>nfsmount_vol</code> , <code>pcmount_vol</code>
0x2007	<p>E_VALIEN: Wrong volume format. The volume to be mounted is not of the correct. Either it is the wrong type, i.e. mounting an MS-DOS volume with <code>mount_vol()</code>, or it has not been formatted by the pHILE+ file system manager.</p>	<code>cdmount_vol</code> , <code>mount_vol</code> , <code>pcmount_vol</code> , <code>verify_vol</code>
0x2008	<p>E_MNTED: Volume already mounted. This error condition implies one of the following:</p> <ul style="list-style-type: none"> ■ An attempt was made to mount a device that is already mounted. ■ An attempt was made to initialize a mounted volume. 	<code>cdmount_vol</code> , <code>init_vol</code> , <code>mount_vol</code> , <code>nfsmount_vol</code> , <code>pcinit_vol</code> , <code>pcmount_vol</code>
0x2009	<p>E_MNTOPEN: Cannot unmount volume; files open. An attempt was made to unmount a volume when one or more of its files are still open. All files must be closed before a volume can be unmounted.</p>	<code>unmount_vol</code> , <code>verify_vol</code>

TABLE B-3 pHILE+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x200A	E_DMOUNT: Volume not mounted. Attempted to reference an unmounted volume.	change_dir, create_f, get_fn, lseek_f, make_dir, move_f, open_f, open_fn, read_vol, remove_f, sync_vol, unmount_vol, verify_vol, write_vol
0x200B	E_FNAME: Filename not found. One or more of the filenames specified in a pathname cannot be located. *	change_dir, create_f, get_fn, make_dir, move_f, open_f, remove_f
0x200C	E_IFN: Illegal pathname. The pathname as specified is illegal. Possibilities are: <ul style="list-style-type: none"> ■ File name exceeds 12 characters. ■ Illegal character in a filename. ■ Illegal first character in a filename. ■ Incorrect pathname syntax. * 	access_f, change_dir, chmod_f, chown_f, create_f, get_fn, link_f, lstat_f, make_dir, move_f, open_dir, open_f, open_fn, read_link, remove_f, stat_f, stat_vfs, symlink_f, truncate_f, unmount_vol, utime_f
0x200D	E_NDD: No default directory. A relative path-name has been entered, but the calling task has never done a change_dir() call.	access_f, change_dir, chmod_f, chown_f, create_f, get_fn, link_f, lstat_f, make_dir, move_f, open_dir, open_f, open_fn, read_link, remove_f, stat_f, stat_vfs, symlink_f, truncate_f, unmount_vol, utime_f

TABLE B-3 pHILE+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x200E	<p>E_FORD: Directory file expected. An ordinary file was specified where a directory file was required. Either of the following are possible:</p> <ul style="list-style-type: none"> ■ A file in a pathname (except the last file) is not a directory file. ■ The filename specified on a <code>change_dir()</code> is not a directory file. * 	<code>change_dir</code> , <code>create_f</code> , <code>get_fn</code> , <code>make_dir</code> , <code>move_f</code> , <code>open_f</code> , <code>remove_f</code>
0x200F	E_ASIZ: Illegal Expansion Unit. An expansion unit of zero is illegal.	<code>create_f</code>
0x2010	E_NODE: Null pathname. A pathname with zero characters was passed. This error code is also returned when a pathname that does not end with an actual filename has been passed to <code>create_f()</code> or <code>make_dir()</code> , or to the new filename of a <code>move_f()</code> call. For example, a period (.) would be a legal pathname for <code>open_f()</code> but not for <code>create_f()</code> .	<code>create_f</code> , <code>make_dir</code> , <code>move_f</code> , <code>remove_f</code> ,
0x2011	E_FEXIST: Filename already exists.	<code>create_f</code> , <code>make_dir</code> , <code>move_f</code>
0x2012	E_FLIST: Too many files on volume. Attempt to create a new file when the FLIST is full. You provide the size of the FLIST when the volume is initialized.	<code>create_f</code> , <code>make_dir</code> , <code>move_f</code>
0x2013	E_FOPEN: Cannot remove an open file. Attempt to remove a file that is still open.	<code>remove_f</code>
0x2014	E_DNE: Cannot delete directory that has files. Attempt to remove a directory that is not empty.*	<code>remove_f</code>

TABLE B-3 pHILE+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x2015	<p>E_RO: Requested operation not allowed on this file. This error implies one of the following:</p> <ul style="list-style-type: none"> ■ An attempt was made to write to or lock BITMAP.SYS, FLIST.SYS, or a directory file. ■ Attempted to remove a system file. ■ Attempted to annex to BITMAP.SYS or FLIST.SYS. ■ Attempted to write to a volume mounted with <code>sync_mode</code> set to <code>SM_READ_ONLY</code>. * 	<p><code>annex_f</code>, <code>chmod_f</code>, <code>chown_f</code>, <code>create_f</code>, <code>fchmod_f</code>, <code>fchown_f</code>, <code>ftruncate_f</code>, <code>link_f</code>, <code>lock_f</code>, <code>make_dir</code>, <code>move_f</code>, <code>remove_f</code>, <code>truncate_f</code>, <code>utime_f</code>, <code>write_f</code>, <code>write_vol</code></p>
0x2016	E_DIFDEV: <code>move_f()</code> across volumes. The old and new pathnames specified on a <code>move_f()</code> call are not on the same device.	<code>move_f</code>
0x2017	E_NOTREE: <code>move_f()</code> would destroy directory-tree structure. Attempted to move a directory file to a location within its own sub-tree. If allowed, the volume's file system hierarchy would no longer be a tree structure.	<code>move_f</code>
0x2018	E_OFULL: Too many files open for task. A task attempted to open more files than specified by the pHILE+ Configuration Table parameter <code>fc_ncfile</code> .	<code>open_dir</code> , <code>open_f</code> , <code>open_fn</code> ,
0x2019	E_NOFCB: Too many files open in system. An <code>open_f()</code> attempt will exceed the maximum number of open files allowed in the system as specified by the pHILE+ Configuration Table parameter <code>fc_nfcfb</code> .	<code>open_dir</code> , <code>open_f</code> , <code>open_fn</code> ,
0x201A	<p>E_FIDBIG: Invalid FID, out of range. The FID provided on a pHILE+ call has a value that could not have been returned by an <code>open_f()</code> call.</p> <p><code>close_dir()</code> will return this error code only if <code>dd_fn</code> in the XDIR has been corrupted.</p>	<p><code>annex_f</code>, <code>close_dir</code>, <code>close_f</code>, <code>fchmod_f</code>, <code>fchown_f</code>, <code>fstat_f</code>, <code>fstat_vfs</code>, <code>ftruncate_f</code>, <code>lock_f</code>, <code>lseek_f</code>, <code>read_f</code>, <code>write_f</code></p>

B

TABLE B-3 pHILE+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x201B	E_FIDOFF: Invalid file ID, file not open. The file ID provided on a pHILE+ call is not an open file.	<code>annex_f</code> , <code>close_dir</code> , <code>close_f</code> , <code>fchmod_f</code> , <code>fchown_f</code> , <code>fstat_f</code> , <code>fstat_vfs</code> , <code>ftruncate_f</code> , <code>lock_f</code> , <code>lseek_f</code> , <code>read_f</code> , <code>write_f</code>
0x201C	E_ININFULL: Index block full. The physical size of a file cannot be increased because the file's index block is full, so that no more extent descriptors can be added to the file. This error code indicates that the file is badly scattered across the device. The <code>annex_f()</code> call should be used to produce more contiguity in the file and reduce the number of extents. On NFS volumes this error code is returned if a file is too big.*	<code>annex_f</code> , <code>create_f</code> , <code>ftruncate_f</code> , <code>make_dir</code> , <code>move_f</code> , <code>truncate_f</code> , <code>write_f</code>
0x201D	E_VFULL: Volume full. No more free blocks of the required type (control or data) are available on the volume. This error can occur on a <code>write_f()</code> call whenever the file is extended; on a <code>create_f()</code> , <code>make_dir()</code> , or <code>move_f()</code> if a directory must be extended; or on an <code>annex_f()</code> call. *	<code>annex_f</code> , <code>create_f</code> , <code>ftruncate_f</code> , <code>make_dir</code> , <code>move_f</code> , <code>truncate_f</code> , <code>write_f</code>
0x201E	E_BADPOS: Illegal position parameter to <code>lseek_f()</code> . The <code>LSEEK</code> position parameter must be 0, 1, or 2.	<code>lseek_f</code>
0x201F	E_EOF: Seek past end of file. The parameters provided to <code>lseek_f()</code> would position the <code>L_ptr</code> beyond the logical end of the file. Since the <code>L_ptr</code> is viewed by the pHILE+ file system manager as unsigned, this error can also occur when an <code>lseek_f()</code> call positions the <code>L_ptr</code> before the beginning of a file.	<code>lseek_f</code> , <code>read_dir</code>

TABLE B-3 pHILE+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x2021	E_ILLDEV: Illegal device. The major device number specified in an <code>init_vol()</code> or <code>mount_vol()</code> is larger than the maximum device number specified in the pSOS+ Configuration Table.	<code>cdmount_vol</code> , <code>init_vol</code> , <code>mount_vol</code> , <code>pcmount_vol</code> , <code>verify_vol</code>
0x2022	E_LOCKED: Data is locked. Attempt to access a region of a file that is locked.	<code>ftruncate_f</code> , <code>lock_f</code> , <code>read_f</code> , <code>truncate_f</code> , <code>write_f</code>
0x2023	E_BADFN: Illegal or unused filename. The FN passed to an <code>open_fn()</code> call is not legal. Either it is not within the limits of the FLIST or the corresponding FD is not in use.	<code>open_fn</code>
0x2024	E_FMODE: Bad synchronization mode to <code>mount_vol()</code> . The <code>mount_vol()</code> synchronization mode must be 0, 1, or 2.	<code>cdmount_vol</code> , <code>mount_vol</code> , <code>pcmount_vol</code>
0x2025	E_IDN: Illegal device name. An illegal device name was passed to a function requiring either a device or pathname as input. Either the device number was illegal (i.e., major/minor numbers out of bounds) or it contained a syntax error.	<code>access_f</code> , <code>cdmount_vol</code> , <code>change_dir</code> , <code>chmod_f</code> , <code>chown_f</code> , <code>create_f</code> , <code>get_fn</code> , <code>init_vol</code> , <code>link_f</code> , <code>lstat_f</code> , <code>make_dir</code> , <code>mount_vol</code> , <code>move_f</code> , <code>nfsmount_vol</code> , <code>open_dir</code> , <code>open_f</code> , <code>open_fn</code> , <code>pcinit_vol</code> , <code>pcmount_vol</code> , <code>read_link</code> , <code>read_vol</code> , <code>remove_f</code> , <code>stat_f</code> , <code>stat_vfs</code> , <code>symlink_f</code> , <code>sync_vol</code> , <code>truncate_f</code> , <code>unmount_vol</code> , <code>utime_f</code> , <code>verify_vol</code> , <code>write_vol</code>

TABLE B-3 pHILE+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x2026	E_BADMS: MS-DOS volume; illegal operation. This function cannot be used with MS-DOS volumes.	access_f, annex_f, chmod_f, chown_f, fchmod_f, fchown_f, link_f, lock_f, lstat_f, read_link, symlink_f, utime_f,
0x2027	E_ILLMSTYP: Illegal DOS disk type. The <code>pcinit_vol()</code> parameter <code>dktype</code> exceeds the maximum allowable value.	pcinit_vol
0x2029	E_NMSVOL: Cannot mount MS-DOS volume. The pHILE+ Configuration Table entry <code>fc_msdos</code> is zero, indicating MS-DOS volumes cannot be mounted.	pcmount_vol
0x2041	E_BUFSIZE: Buffers not available for block size.	cdmount_vol
0x2050	E_BADNFS: NFS volume; illegal operation. This function cannot be used with NFS volumes.	annex_f, get_fn, lock_f, open_fn, read_vol, sync_vol, write_vol
0x2051	E_MAXLOOP: Symbolic links nested too deeply. A pathname contains symbolic links nested more than three levels deep.	access_f, cdmount_vol, change_dir, chmod_f, chown_f, create_f, link_f, lstat_f, make_dir, move_f, nfsmount_vol, open_dir, open_f, pcinit_vol, pcmount_vol, read_link, remove_f, stat_f, stat_vfs, symlink_f, truncate_f, unmount_vol, utime_f, verify_vol

TABLE B-3 pHILE+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x2052	E_REMOTE: "Too many levels of remote in path" on server. *	access_f, change_dir, chmod_f, chown_f, create_f, fchmod_f, fchown_f, fstat_f, fstat_vfs, ftruncate_f, link_f, lseek_f, lstat_f, make_dir, move_f, nfsmount_vol, open_dir, open_f, read_dir, read_f, read_link, remove_f, stat_f, stat_vfs, symlink_f, truncate_f, unmount_vol, utime_f, write_f
0x2053	E_PERM: The task does not have the ownership that is needed. The task does not have ownership for the requested file operation. *	remove_f
0x2054	E_EIO: A hard error occurred at a remote site. There was some hardware error, such as an I/O error, at the server. Abort the operation. *	access_f, change_dir, chmod_f, chown_f, create_f, fchmod_f, fchown_f, fstat_f, fstat_vfs, ftruncate_f, link_f, lseek_f, lstat_f, make_dir, move_f, nfsmount_vol, open_dir, open_f, read_dir, read_f, read_link, remove_f, stat_f, stat_vfs, symlink_f, truncate_f, unmount_vol, utime_f, write_f

B

TABLE B-3 pHILE+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x2055	E_EACCES: The task does not have the necessary access permissions. The task does not have permission for the requested file operation. *	access_f, change_dir, chmod_f, chown_f, close_dir, create_f, fchmod_f, fchown_f, fstat_f, fstat_vfs, ftruncate_f, link_f, lseek_f, lstat_f, make_dir, move_f, nfsmount_vol, open_dir, open_f, read_dir, read_f, read_link, remove_f, stat_f, stat_vfs, symlink_f, truncate_f, utime_f, write_f
0x2056	E_EISDIR: Illegal operation on a directory. If you attempt an operation on a directory as if it were a data file, this error is reported.*	move_f, read_f, remove_f, write_f
0x2057	E_EQUOT: Quota exceeded. The server enforces a disk usage quota for each user. If this error is reported, use the disk less, or remove files, or have the quota raised. *	create_f, ftruncate_f, link_f, make_dir, move_f, symlink_f, truncate_f, write_f

TABLE B-3 pHILE+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x2058	E_ESTALE: Stale file handle, file handle invalid. When a server crashes, or there is some other exceptional event, file handles no longer are valid. Consider unmounting the file system and remounting it. *	access_f, change_dir, chmod_f, chown_f, create_f, fchmod_f, fchown_f, fstat_f, fstat_vfs, ftruncate_f, link_f, lseek_f, lstat_f, make_dir, move_f, nfsmount_vol, open_dir, open_f, read_dir, read_f, read_link, remove_f, stat_f, stat_vfs, symlink_f, truncate_f, unmount_vol, utime_f, write_f
0x2059	E_XLINK: Can't close link.	link_f
0x205A	E_NAMETOOLONG: Directory/filename too long.	read_dir
0x205B	E_ENXIO: "No such device or address" on server.*	access_f, change_dir, chmod_f, chown_f, create_f, fchmod_f, fchown_f, fstat_f, fstat_vfs, ftruncate_f, link_f, lseek_f, lstat_f, make_dir, move_f, nfsmount_vol, open_dir, open_f, read_dir, read_f, read_link, remove_f, stat_f, stat_vfs, symlink_f, truncate_f, unmount_vol, utime_f, write_f

B

TABLE B-3 pHILE+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x205C	E_ENODEV: "No such device" on server. *	access_f, change_dir, chmod_f, chown_f, create_f, fchmod_f, fchown_f, fstat_f, fstat_vfs, ftruncate_f, link_f, lseek_f, lstat_f, make_dir, move_f, nfsmount_vol, open_dir, open_f, read_dir, read_f, read_link, remove_f, stat_f, stat_vfs, symlink_f, truncate_f, unmount_vol, utime_f, write_f
0x2060	E_BADCD: CD-ROM volume; illegal operation.	access_f, annex_f, chmod_f, chown_f, create_f, fchmod_f, fchown_f, ftruncate_f, link_f, lock_f, lstat_f, make_dir, move_f, read_link, remove_f, symlink_f, sync_vol, truncate_f, utime_f, write_f, write_vol
0x2061	E_NCDVOL: Not configured for CD-ROM volumes.	cdmount_vol
0x2062	E_CDMVOL: Multi-volume CD-ROM not supported.	cdmount_vol
0x2063	E_CDBSIZE: Volume not made with 2K block size.	cdmount_vol
0x2064	E_CDFMT: CD format not ISO 9660 compatible.	cdmount_vol

TABLE B-3 pHILE+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x2070	E_EAUTH: "Authentication error" on server. *	access_f, change_dir, chmod_f, chown_f, create_f, fchmod_f, fchown_f, fstat_f, fstat_vfs, ftruncate_f, link_f, lseek_f, lstat_f, make_dir, move_f, nfsmount_vol, open_dir, open_f, read_dir, read_f, read_link, remove_f, stat_f, stat_vfs, symlink_f, truncate_f, unmount_vol, utime_f, write_f
0x2071	E_ENFS: NFS error. "Portmap error" on server.*	access_f, change_dir, chmod_f, chown_f, create_f, fchmod_f, fchown_f, fstat_f, fstat_vfs, ftruncate_f, link_f, lseek_f, lstat_f, make_dir, move_f, nfsmount_vol, open_dir, open_f, read_dir, read_f, read_link, remove_f, stat_f, stat_vfs, symlink_f, truncate_f, unmount_vol, utime_f, write_f

B

TABLE B-3 pHILE+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x2072	E_ETIMEDOUT: NFS call timed out. A server did not respond to a request. The requested NFS call timed out. Network congestion, a server that is down, or a hardware problem may be the cause.*	access_f, change_dir, chmod_f, chown_f, create_f, fchmod_f, fchown_f, fstat_f, fstat_vfs, ftruncate_f, link_f, lseek_f, lstat_f, make_dir, move_f, nfsmount_vol, open_dir, open_f, read_dir, read_f, read_link, remove_f, stat_f, stat_vfs, symlink_f, truncate_f, unmount_vol, utime_f, write_f
0x2074	E_ENOAUTHBLK: No RPC authorization blocks available.*	access_f, change_dir, chmod_f, chown_f, create_f, fchmod_f, fchown_f, fstat_f, fstat_vfs, ftruncate_f, link_f, lseek_f, lstat_f, make_dir, move_f, nfsmount_vol, open_dir, open_f, read_dir, read_f, read_link, remove_f, stat_f, stat_vfs, symlink_f, truncate_f, unmount_vol, utime_f, write_f

TABLE B-3 pHILE+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x2075	E_ECANTSEND: Failure in sending call. *	access_f, change_dir, chmod_f, chown_f, create_f, fchmod_f, fchown_f, fstat_f, fstat_vfs, ftruncate_f, link_f, lseek_f, lstat_f, make_dir, move_f, nfsmount_vol, open_dir, open_f, read_dir, read_f, read_link, remove_f, stat_f, stat_vfs, symlink_f, truncate_f, umount_vol, utime_f, write_f
0x2076	E_ECANTRECV: Failure in receiving result. *	access_f, change_dir, chmod_f, chown_f, create_f, fchmod_f, fchown_f, fstat_f, fstat_vfs, ftruncate_f, link_f, lseek_f, lstat_f, make_dir, move_f, nfsmount_vol, open_dir, open_f, read_dir, read_f, read_link, remove_f, stat_f, stat_vfs, symlink_f, truncate_f, umount_vol, utime_f, write_f

B

TABLE B-3 pHILE+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x2077	E_PROGUNAVAIL: Program not available. *	access_f, change_dir, chmod_f, chown_f, create_f, fchmod_f, fchown_f, fstat_f, fstat_vfs, ftruncate_f, link_f, lseek_f, lstat_f, make_dir, move_f, nfsmount_vol, open_dir, open_f, read_dir, read_f, read_link, remove_f, stat_f, stat_vfs, symlink_f, truncate_f, unmount_vol, utime_f, write_f
0x2078	E_EPROGVERSMISMATCH: Program version mismatched. *	access_f, change_dir, chmod_f, chown_f, create_f, fchmod_f, fchown_f, fstat_f, fstat_vfs, ftruncate_f, link_f, lseek_f, lstat_f, make_dir, move_f, nfsmount_vol, open_dir, open_f, read_dir, read_f, read_link, remove_f, stat_f, stat_vfs, symlink_f, truncate_f, unmount_vol, utime_f, write_f

TABLE B-3 pHILE+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x2079	E_ECANTDECODEARGS: Decode arguments error. *	access_f, change_dir, chmod_f, chown_f, create_f, fchmod_f, fchown_f, fstat_f, fstat_vfs, ftruncate_f, link_f, lseek_f, lstat_f, make_dir, move_f, nfsmount_vol, open_dir, open_f, read_dir, read_f, read_link, remove_f, stat_f, stat_vfs, symlink_f, truncate_f, unmount_vol, utime_f, write_f
0x207A	E_EUNKNOWNHOST: Unknown host name. *	access_f, change_dir, chmod_f, chown_f, create_f, fchmod_f, fchown_f, fstat_f, fstat_vfs, ftruncate_f, link_f, lseek_f, lstat_f, make_dir, move_f, nfsmount_vol, open_dir, open_f, read_dir, read_f, read_link, remove_f, stat_f, stat_vfs, symlink_f, truncate_f, unmount_vol, utime_f, write_f

B

TABLE B-3 pHILE+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x207B	E_EPROGNOTREGISTERED: Remote program is not registered. *	access_f, change_dir, chmod_f, chown_f, create_f, fchmod_f, fchown_f, fstat_f, fstat_vfs, ftruncate_f, link_f, lseek_f, lstat_f, make_dir, move_f, nfsmount_vol, open_dir, open_f, read_dir, read_f, read_link, remove_f, stat_f, stat_vfs, symlink_f, truncate_f, unmount_vol, utime_f, write_f
0x207C	E_UNKNOWNPROTO: Unknown protocol. *	access_f, change_dir, chmod_f, chown_f, create_f, fchmod_f, fchown_f, fstat_f, fstat_vfs, ftruncate_f, link_f, lseek_f, lstat_f, make_dir, move_f, nfsmount_vol, open_dir, open_f, read_dir, read_f, read_link, remove_f, stat_f, stat_vfs, symlink_f, truncate_f, unmount_vol, utime_f, write_f

TABLE B-3 pHILE+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x207D	E_EINTR: Call interrupted. *	access_f, change_dir, chmod_f, chown_f, create_f, fchmod_f, fchown_f, fstat_f, fstat_vfs, ftruncate_f, link_f, lseek_f, lstat_f, make_dir, move_f, nfsmount_vol, open_dir, open_f, read_dir, read_f, read_link, remove_f, stat_f, stat_vfs, symlink_f, truncate_f, unmount_vol, utime_f, write_f
0x207E	E_ERPC: All other RPC errors. *	access_f, change_dir, chmod_f, chown_f, create_f, fchmod_f, fchown_f, fstat_f, fstat_vfs, ftruncate_f, link_f, lseek_f, lstat_f, make_dir, move_f, nfsmount_vol, open_dir, open_f, read_dir, read_f, read_link, remove_f, stat_f, stat_vfs, symlink_f, truncate_f, unmount_vol, utime_f, write_f
0x2200	VF_INSUFF: Insufficient working area provided. Supply more memory for the data area pointed to by pb_dataptr and increase pb_dataalen. Refer to page 2-127.	verify_vol

TABLE B-3 pHILE+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x2201	VF_MAXDEPTH: Maximum depth exceeded on directory traversal. Increase the value of pb_maxdepth. If needed, supply more memory for pb_dataptr and increase pb_dataLEN. Refer to page 2-127.	verify_vol
0x2202	VF_ABORT: Verify routine aborted by user.	verify_vol
0x2F01	FAT_NORAM: Insufficient data area.	This error originates in pHILE+ initialization.
0x2F0E	FAT_PHCSUM: Checksum error in the pHILE+ file system manager.	This error originates in pHILE+ initialization.

B.2.1 pSOS+ Errors Related to pHILE+

When a task is deleted or restarted, pSOS+ may return errors related to pHILE+. These error codes are listed in Table B-4.

TABLE B-4 pSOS+ Errors Related to pHILE+

Hex	Description	System Call(s)
0x0D	ERR_RSTFS: Possible file system corruption. A task was restarted while executing pHILE+ code, resulting in a possible inconsistency in the volume data structures. Following this error, use of verify_vol() on the volume is recommended.	(pSOS+) t_restart
0x18	ERR_DELFs: Attempt to delete task using the pHILE+ file system manager. A task that has open files or is holding pHILE+ resources cannot be deleted.	(pSOS+) t_delete

B.2.2 Conversions of NFS Error Codes

All NFS errors received by pHILE+ are mapped to pHILE+ error codes. Table B-5 shows the conversions of these codes. If an NFS error not listed below is received, it is mapped to the code `E_FAIL 0x2002`, “pHILE+ failure”. This should never happen, unless a new NFS error code is defined at the server.

Codes whose second-to-last digit is not 5 can also represent errors from other file systems.

TABLE B-5 pHILE+ Error Codes That Represent NFS Errors

pHILE+ Hex	pHILE+ Description	NFS Hex	NFS Description
0x200B	<code>E_FNAME</code> : Filename not found.	0x02	No such file or directory.
0x200C	<code>E_IFN</code> : Illegal pathname.	0x3f	File name too long.
0x200E	<code>E_FORD</code> : Directory file expected.	0x14	Not a directory.
0x2011	<code>E_FEXIST</code> : File already exists.	0x11	File exists.
0x2014	<code>E_DNE</code> : Directory not empty.	0x42	Directory not empty.
0x2015	<code>E_RO</code> : Illegal on system or directory file.	0x1e	Read-only file system.
0x201C	<code>E_ININFULL</code> : Index block full.	0x1b	File too large.
0x201D	<code>E_VFULL</code> : Volume is full.	0x1c	No space left on device.
0x2052	<code>E_EREMOTE</code> : Too many levels of remote in path.	0x47	Too many levels of remote in path.
0x2053	<code>E_PERM</code> : Task does not have ownership.	0x01	Not owner.
0x2054	<code>E_EIO</code> : Hard error happened at remote site.	0x05	I/O error.
0x2055	<code>E_EACCESS</code> : Task does not have access permissions.	0x0d	Permission denied.
0x2056	<code>E_EISDIR</code> : Illegal operation on a directory.	0x15	Is a directory.

TABLE B-5 pHILE+ Error Codes That Represent NFS Errors

pHILE+ Hex	pHILE+ Description	NFS Hex	NFS Description
0x2057	E_QUOT: Quota exceeded.	0x45	Disc quota exceeded.
0x2058	E_STALE: Stale NFS file handle.	0x46	Stale NFS file handle.
0x205B	E_ENXIO: No such device or address.	0x06	No such device or address.
0x205C	E_ENODEV: No such device.	0x13	No such device.

B.2.3 Conversions of RPC Error Codes

All RPC errors received by pHILE+ are mapped to pHILE+ error codes. Table B-6 shows the conversions of these codes. If an RPC error code not listed below is received, it is mapped to the code E_ERPC 0x207E, "All other RPC errors".

TABLE B-6 pHILE+ Error Codes That Represent RPC Errors

pHILE+ Hex	pHILE+ Description	RPC Code	RPC Description
0x2070	E_EAUTH: RPC Authorization is not available.	7	RPC_AUTHERROR: Authentication error.
0x2071	E_ENFS: NFS error - pmap failure.	14	RPC_PORTMAPFAILURE: The pmapper failed in its call.
0x2072	E_ETIMEDOUT: NFS call timed out.	5	RPC_TIMEDOUT: Call timed out.
0x2075	E_ECANTSEND: Failure in sending call.	3	RPC_CANTSEND: Failure in sending call.
0x2076	E_ECANTRECV: Failure in receiving result.	4	RPC_CANTRECV: Failure in receiving result.
0x2077	E_PROBUNAVAIL: Program not available.	8	RPC_PROGUNAVAIL: Program not available.
0x2078	E_PROGVERSISMATCH: Program version mismatched.	9	RPC_PROGVERSISMATCH: Program version mismatched.
0x2079	E_ECANTDECODEARGS: Decode arguments error.	11	RPC_CANTDECODEARGS: Decode arguments error.

TABLE B-6 pHILE+ Error Codes That Represent RPC Errors

pHILE+ Hex	pHILE+ Description	RPC Code	RPC Description
0x207A	E_EUNKNOWNHOST: Unknown host name.	13	RPC_UNKNOWNHOST: Unknown host name.
0x207B	E_PROGNOTREGISTERED: Remote program is not registered.	15	RPC_PROGNOTREGISTERED: Remote program is not registered.
0x207C	E_UNKNOWNPROTO: Unknown protocol.	17	RPC_UNKNOWNPROTO: Unknown protocol.
0x207D	E_EINTR: Call interrupted.	18	RPC_INTR: Call interrupted.
0x207E	E_ERPC: All other RPC errors.	1	RPC_CANTENCODEARGS: Can't encode arguments.
		2	RPC_CANTDECODERES: Can't decode results.
		6	RPC_VERSMISMATCH: RPC versions not compatible.
		10	RPC_PROCUNAVAIL: Procedure unavailable.
		12	RPC_SYSTEMERROR: Generic "other problem".
		16	RPC_FAILED

B

B.3 pREPC+ Error Codes

When a pREPC+ system call generates an error, an error code is loaded into an internal variable that can be read through the macro `errno()`. One `errno` variable exists for each task. If the return value of a pREPC+ system call indicates an error, your application should examine the `errno` variable to determine the cause of the error. See the description of `errno()` on page 3-27 for more information.

Table B-7 lists the error codes of the pREPC+ library and component. Each listing includes the error code's hexadecimal number, its mnemonic, and a brief description. The error code mnemonics are also defined in `<prepc.h>`.

For practical reasons, system calls are not listed, because nearly every pREPC+ error code can be returned by all pREPC+ system calls. In addition, errors in other pSOSystem components or device drivers can be reported by pREPC+ system calls.

TABLE B-7 pREPC+ Error Codes

Hex	Mnemonic and Description
0x3001	EMOPEN: Maximum number of files are open.
0x3002	ERANGE: Converted value out of range.
0x3003	EBASE: Invalid radix base specified.
0x3005	EACCESS: File access violation.
0x3006	EMODE: Unrecognized mode specified.
0x3007	EINVAL: Operation not allowed on this type of file.
0x3008	EPHILE: Attempted a disk file operation without the PHILE+ file system manager installed.
0x3009	EINVTYPE: Invalid buffer type.
0x300a	EINVSIZE: Invalid buffer size.
0x300b	EPRRW: Previous read/write; cannot setvbuf.
0x300d	ENAN: Invalid floating point number.
0x3F01	LC_FAT_CONFIG: Insufficient memory to hold pREPC+ data.
0x3F03	LC_FAT_STDIO: Cannot open standard I/O streams.
0x3F0E	LC_FAT_CHKSUM: Corrupted ROM; checksum error.

B.4 pNA+ Error Codes

When the pNA+ network manager generates an error, an error code is loaded into an internal variable that can be read through the macro `errno()`. One `errno` variable exists for each task. If the return value of a pNA+ system call indicates an error, your application should examine the `errno` variable to determine the cause of the error. See the description of `errno()` on page 3-27 for more information.

Table B-8 lists the error codes of the pNA+ network manager. Each listing includes the error code's hexadecimal number, its mnemonic and description, and the system calls that can return it. The error code mnemonics are also defined in the file `<pna.h>`.

TABLE B-8 pNA+ Error Codes

Hex	Mnemonic and Description	System Call(s)
0x5006	ENXIO: No such address.	ioctl
0x5009	EBADS: The socket descriptor is invalid.	accept, bind, close, connect, getpeername, getsockname, getsockopt, ioctl, recv, recvfrom, recvmsg, select, send, sendmsg, sendto, setsockopt, shr_socket, shutdown
0x500D	EACCESS: Permission denied.	send, sendmsg, sendto
0x5011	EEXIST: Duplicate entry exists.	ioctl

B

TABLE B-8 pNA+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x5016	EINVAL: An argument is invalid.	accept, add_ni, bind, chng_route, close, connect, ioctl, listen, recv, recvfrom, recvmsg, send, sendmsg, sendto, setsockopt, shutdown, socket
0x5017	ENFILE: An internal table has run out of space.	accept, add_ni, chng_route, shr_socket, socket
0x5020	EPIPE: The connection is broken.	send, sendmsg, sendto
0x5023	EWOULDBLOCK: This operation would block, but socket is non-blocking.	accept, recv, recvfrom, recvmsg, send, sendmsg, sendto
0x5024	EINPROGRESS: The socket is non-blocking, and the connection cannot be completed immediately.	connect
0x5025	EALREADY: The socket is non-blocking, and a previous connection attempt has not yet been completed.	connect
0x5027	EDESTADDRREQ: The destination address is invalid.	sendmsg, sendto
0x5028	EMSGSIZE: Message too long.	recvmsg, send, sendmsg, sendto
0x5029	EPROTOTYPE: Wrong protocol type for socket.	socket
0x502A	ENOPROTOOPT: Protocol not available.	getsockopt, setsockopt
0x502B	EPROTONOSUPPORT: Protocol not supported.	socket

TABLE B-8 pNA+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x502D	EOPNOTSUPP: Requested operation not valid for this type of socket.	accept, ioctl, listen
0x502F	EAFNOSUPPORT: Address family not supported.	connect
0x5030	EADDRINUSE: Address is already in use.	bind
0x5031	EADDRNOTAVAIL: Address not available.	bind, connect, sendmsg, sendto
0x5033	ENETUNREACH: Network is unreachable.	chng_route, send, sendmsg, sendto
0x5035	ECONNABORTED: The connection has been aborted by the peer.	accept
0x5036	ECONNRESET: The connection has been reset by the peer.	recv, recvfrom, recvmsg, send, sendmsg, sendto
0x5037	ENOBUFS: An internal buffer is required but cannot be allocated.	connect, getpeername, getsockname, ioctl, send, sendmsg, sendto, setsockopt, socket
0x5038	EISCONN: The socket is already connected.	connect, sendmsg, sendto
0x5039	ENOTCONN: The socket is not connected.	getpeername, recv, recvfrom, recvmsg, send, sendmsg, sendto, shutdown
0x503B	ETOOMANYREFS: Too many references: can't splice.	setsockopt
0x503C	ETIMEDOUT: Connection timed out.	connect
0x503D	ECONNREFUSED: The attempt to connect was refused.	connect, listen

B

TABLE B-8 pNA+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x5041	EHOSTUNREACH: The destination host could not be reached from this node.	send, sendto, sendmsg
0x5046	ENIDOWN: NI_INIT returned -1.	add_ni
0x5047	ENMTU: The MTU is invalid.	add_ni
0x5048	ENHWL: The <i>hardware</i> length is invalid.	add_ni
0x5049	ENNOFIND: The route specified cannot be found.	chnng_route
0x504A	ECOLL: Collision in select call; these conditions have already been selected by another task.	select
0x504B	ETID: The task ID is invalid.	ioctl, set_id, shr_socket
0x5F01	FAT_INSUFFMEM: Insufficient memory allocated by nc_datasize or Region 0 too small; increase nc_datasize of Region 0 or reduce the number of required data structures specified in the pNA+ Configuration Table.	This error originates in pNA+ initialization.
0x5F02	FAT_NRT: The number of initial routing table entries specified exceeds nc_nroute. Increase nc_nroute.	This error originates in pNA+ initialization.
0x5F03	FAT_NNI: The number of initial NI table entries specified exceeds nc_nni. Increase nc_nni.	This error originates in pNA+ initialization.
0x5F04	FAT_NIHSIZE: Invalid NI address.	This error originates in pNA+ initialization.
0x5F05	FAT_NIMTU: Invalid MTU for NI.	This error originates in pNA+ initialization.
0x5F06	FAT_PNAMEM: pNA+ memory error.	This error originates in pNA+ initialization.
0x5F07	FAT_PNATASK: PNAD task creation error.	This error originates in pNA+ initialization.
0x5F08	FAT_PNAINIT: pNA+ initialization error.	This error originates in pNA+ initialization.

TABLE B-8 pNA+ Error Codes (Continued)

Hex	Mnemonic and Description	System Call(s)
0x5F09	FAT_NIINIT: NI initialization error.	This error originates in pNA+ initialization.
0x5F0A	FAT_RTINIT: Routing table initialization error.	This error originates in pNA+ initialization.
0x5F0B	FAT_ARPINIT: ARP table initialization error.	This error originates in pNA+ initialization.
0x5F0C	FAT_TIMERINIT: PNAD timer initialization error.	This error originates in pNA+ initialization.
0x5F0D	FAT_EVENT: PNAD event error.	This error originates in pNA+ initialization.
0x5F0E	FAT_CHKSUM: pNA+ checksum error.	This error originates in pNA+ initialization.
0x5F0F	EINTERNAL: pNA+ detects an inconsistency in the control information for network resources it manages. In most cases, this error is caused by a corruption of the pNA+ data area by an errant user task. Check pNA+ data configurations and memory usage by tasks. For further advice, contact pSOSystem technical support with information on pNA+ configuration and a dump of register contents.	This error can be detected at several points within pNA+.
0x5F10	FAT_NHT: The number of initial host table entries specified exceeds nc_nhentry. Increase nc_nhentry.	This error originates in pNA+ initialization.
0x5F11	FAT_FUNC: An invalid system call code was passed to pNA+. Check the pNA bindings file in pSOSystem.	This error originates when a system call is made.

B

B.5 pRPC+ Error Codes

When a pRPC+ system call generates an error, an error code is loaded into an internal variable that can be read through the macro `errno()`. One `errno` variable exists for each task. If the return value of a pRPC+ system call indicates an error, your application should examine the `errno` variable to determine the cause of the error. See the description of `errno()` on page 3-27 for more information.

Table B-9 lists the error codes of the pRPC+ subcomponent. Each listing includes the error code's hexadecimal number, its mnemonic and description, and the system calls that can return it. The error code mnemonics are also defined in `<prpc.h>`.

TABLE B-9 pRPC+ Error Codes

Hex	Mnemonic and Description	System Call(s)
0x5101	FAT_PRPC_CHKSUM: Corrupted ROM.	This error originates in pRPC+ initialization.
0x5102	FAT_PRPC_MEM: Insufficient memory to hold pRPC+ data.	This error originates in pRPC+ initialization.
0x5104	FAT_PRPC_TASKCREATE: Cannot start pmap.	This error originates in pRPC+ initialization.

B.6 Driver Error Codes

The tables below list the error codes returned by pSOSystem drivers. Each driver's table lists all the error codes returned by that driver. Error code information includes hexadecimal numbers, mnemonics and descriptions.

Drivers return error codes through the pSOS+ Kernel-to-Driver Interface using the `out_retval` element of the `ioparms` structure. The contents of `out_retval` are copied to the variable pointed to by the service call input parameter `retval`. The parameter `retval` is part of the Application-to-pSOS+ Interface. For example, any driver errors resulting from a `de_read()` service call to a driver are returned at the address pointed to by the `retval` argument using the following syntax:

```
err_code = de_read(dev, iopb, &retval);
```

See the chapter “I/O System” in *pSOSystem System Concepts* for more information on the pSOS+ Kernel-to-Driver Interface and the Application-to-pSOS+ Interface.

Aside from the service calls `de_init()`, `de_open()`, `de_close()`, `de_read()`, `de_write()`, and `de_cntrl()`, a system call can return a driver error code directly through the return value of the system call. For example, the `pHILE+ write_f()` system call can return the SCSI driver error code `SCSI_W_PROTECTED` (0x1050001A) if a write is attempted on a write-protected drive.

B.6.1 Shared Memory Network Interface Driver Error Codes

The error codes listed in Table B-10 are returned by the Shared Memory Network Interface (NI_SMEM) driver.

TABLE B-10 Shared Memory Network Interface Driver Error Codes

Hex	Mnemonic and Description
0x10000001	NISMEM_FAT_IPA: Invalid IP address.

B.6.2 Shared Memory Kernel Interface Driver Error Codes

The error codes listed in Table B-11 are returned by the Shared Memory Kernel Interface (KI_SMEM) driver.

TABLE B-11 Shared Memory Kernel Interface Driver Error Codes

Hex	Mnemonic and Description
0x10000101	KISMEM_FAT_NOPB: No packet buffers.
0x10000102	KISMEM_FAT_NOQ: No queue.
0x10000103	KISMEM_FAT_NOTSUPP: Service not supported.
0x10000104	KISMEM_FAT_BINSTALL: Can't install bus error handler.
0x10000105	KISMEM_FAT_NOD: Number of nodes greater than maximum.



B.6.3 Terminal Interface Driver Error Codes

The error codes listed in Table B-12 are returned by the terminal interface driver.

TABLE B-12 Terminal Interface Driver Error Codes

Hex	Mnemonic and Description
0x10010200	TERM_HDWR: Hardware error.
0x10010201	TERM_MINOR: Invalid minor device.
0x10010203	TERM_BAUD: Invalid baud rate.
0x10010204	TERM_NINIT: Driver not initialized.
0x10010205	TERM_DATA: Cannot allocate data area.
0x10010206	TERM_SEM: Semaphore error.
0x10010210	TERM_AINIT: Console already initialized.
0x10010211	TERM_CHARSIZE: Bad character size.
0x10010212	TERM_BADFLAG: Flag not defined.
0x10010213	TERM_NHWFC: Hardware flow not supported.
0x10010214	TERM_BRKINT: Terminated by a break character.
0x10010215	TERM_DCDINT: Terminated by a loss of DCD.
0x10010216	TERM_NBUFF: No buffers to copy characters (allocb failed).
0x10010217	TERM_NOPEN: Minor device not opened.
0x10010218	TERM_AOPEN: Channel already opened.
0x10010219	TERM_ADOPEN: Channel already opened by another driver.
0x10010220	TERM_CFGHSUP: Hardware does not support channel as configured.
0x10010221	TERM_OUTSYNC: Out of sync with DISI.
0x10010222	TERM_BADMIN: MinChar is greater than RBufSize.
0x10010223	TERM_LDERR: Lower driver error may be corrupted structure.
0x10010224	TERM_QUE: Queue error.
0x10010225	TERM_RXERR: Data receive error.

TABLE B-12 Terminal Interface Driver Error Codes (Continued)

Hex	Mnemonic and Description
0x10010226	TERM_TIMEOUT: Timer expired for read or write.
0x10010227	TERM_CANON: CANON and MinChar and/or MaxTime set (can only have CANON or have MinChar and/or MaxTime).
0x10010228	TERM_ROPER: Redirect operation error.
0x10010229	TERM_MARK: Received a SIOCMARK.
0x10010230	TERM_FRAMING: Framing error.
0x10010231	TERM_PARITY: Parity error.
0x10010232	TERM_OVERRUN: Overrun error.
0x10010233	TERM_NMBLK: No buffer headers (esballoc failed).
0x10010234	TERM_TXQFULL: Transmit queue is full (is returned only if WNWAIT is set).
0x10010235	TERM_NWNCONF: MaxWTime and WNWAIT both set.
0x10010236	TERM_BADCONSL: Bad default console number.
0x10010237	TERM_WABORT: Write was aborted.

B.6.4 Tick Timer Driver Error Codes

The error codes listed in Table B-13 are returned by the tick timer driver.

TABLE B-13 Tick Timer Driver Error Codes

Hex	Mnemonic and Description
0x10020001	TIMR_TICKRATE: Unsupported rate for kc_ticks2sec.

B.6.5 RAM Disk Driver Error Codes

The error codes listed in Table B-14 are returned by the RAM disk driver.

TABLE B-14 RAM Disk Driver Error Codes

Hex	Mnemonic and Description
0x10040001	RDSK_BLOCK: Block number too large.
0x10040002	RDSK_SEM: Semaphore error.
0x10040003	RDSK_MEM: Memory error.

B.6.6 TFTP Driver Error Codes

The error codes listed in Table B-15 are returned by the TFTP driver.

TABLE B-15 TFTP Driver Error Codes

Hex	Mnemonic and Description
0x10060001	TFTP_PROTO: Protocol error detected, such as receipt of a non-DATA packet or lack of an expected message acknowledgment.
0x10060002	TFTP_TMOUT: TFTP server timed out while waiting for a response from the TFTP client.
0x10060003	TFTP_SYNC: TFTP server out of sync with TFTP client.
0x10060004	TFTP_NOSPC: No more free socket IDs.
0x10060005	TFTP_INVALID: Channel number (minor number) exceeds the maximum number of channels the driver can open.
0x10060006	TFTP_NOINIT: Call failed because the TFTP driver has not been initialized and must be initialized before the call can be made.

B.6.7 IDE Driver Error Codes

The error codes listed in Table B-16 are returned by the IDE driver.

TABLE B-16 IDE Driver Error Codes

Hex	Mnemonic and Description
0x10090001	IDE_HDWR: Hardware error.
0x10090002	IDE_MINOR: Invalid minor device.
0x10090003	IDE_CTRL: Invalid function code for IDE_Ctrl().
0x10090004	IDE_NINIT: Device not initialized.
0x10090005	IDE_DATA: Unable to allocate driver data area.
0x10090006	IDE_SEM: Semaphore error.
0x10090007	IDE_BBLK: Bad block.
0x10090008	IDE_UCOR: Uncorrectable error.
0x10090009	IDE_SNF: Sector not found.
0x1009000a	IDE_TONF: Track 0 not found.
0x1009000b	IDE_NDAM: Data address mark not found.
0x1009000c	IDE_RANGE: Block range error.
0x1009F000	IDE_DRV: Drive-related error in the last byte.

B

B.6.8 FLP Driver Error Codes

The error codes listed in Table B-17 are returned by the FLP driver.

TABLE B-17 FLP Driver Error Codes

Hex	Mnemonic and Description
0x100A0001	FLP_MINOR: Invalid minor device.
0x100A0002	FLP_NINIT: Device not initialized.
0x100A0003	FLP_SEM: Semaphore error.
0x100A0004	FLP_QUEUE: Cannot create a message queue.

I

TABLE B-17 FLP Driver Error Codes (Continued)

Hex	Mnemonic and Description
0x100A0005	FLP_TASK: Cannot create the motor control task.
0x100A0006	FLP_RD: Floppy drive read failure.
0x100A0007	FLP_WR: Floppy drive write failure.
0x100A0008	FLP_DATA: Unable to allocate driver data area.
0x100AF000	FLP_DRV: Drive-related error in the last byte.

B.6.9 SCSI Driver Error Codes

The error codes listed in Table B-18 are returned by the SCSI driver.

TABLE B-18 SCSI Error Codes

Hex	Mnemonic and Description
0x10500003	SCSI_FSC: Failed to send SCSI command.
0x10500009	SCSI_CHP: Failed to init SCSI chip.
0x1050000B	SCSI_UKC: Unknown de_ctnrl() function.
0x1050000C	SCSI_ID_ERR: Bad SCSI ID de_ctnrl().
0x1050000D	SCSI_NULL_CDB: NULL SCSI Control block.
0x1050000E	SCSI_PTR_CONFLICT: Both in and out data length given.
0x1050000F	SCSI_DATA_PTR_NULL: NULL data pointer.
0x10500010	SCSI_NOT_INIT: SCSI Driver not initialized.
0x10500011	SCSI_ILLRECON: Bad reconnection (no disconnect).
0x10500012	ESDNOTTDIR: Incorrect device type for operation.
0x10500013	ESDNODEVICE: No such device on SCSI bus.
0x10500014	ESBLOCKOUTOFRANGE: Block given is beyond end of disk.
0x10500015	ESODDBLOCK: Block size is less than physical.
0x10500016	ESNO_CAPACITY: Capacity shows 0 (floppy not in drive).
0x10500017	SCSI_FORMAT_FAILED: Format command failed.

TABLE B-18 SCSI Error Codes (Continued)

Hex	Mnemonic and Description
0x10500018	SCSI_PART_NUM_BAD: Bad partition number.
0x10500019	SCSI_NOT_PARTITION: Device not partitioned.
0x1050001A	SCSI_W_PROTECTED: Device is write-protected.
0x1050001B	SCSI_NO_MEM: Need memory to complete command not available.
0x1050001C	SCSI_NOT_OPEN: SCSI device not open.
0x1050001D	ESDALLREADYOPEN: SCSI device is already open.
0x1050001E	SCSI_END_OF_FILE: End of tape file encountered.
0x10510000	SCSI_ERR: General SCSI error code SCSI_ERR will be OR-ed with actual SCSI error code.
0x10510001	STAT_CHECKCOND: Target wants to give some info.
0x10510002	STAT_ERR: SCSI error that may be retried.
0x10510003	STAT_TIMEOUT: Target selection timed out.
0x10510004	STAT_BUSY: Target busy try again.
0x10510005	STAT_SEMFAIL: Semaphore call failed.
0x10510006	STAT_NOMEM: No memory available for request.
0x10510007	STAT_RETRYEXC: Failed after allotted retries.
0x10510008	STAT_RESET: SCSI bus reset (should retry).
0x10510009	STAT_BADSIZE: Drive shows no blocks.
0x1051000A	STAT_NOMEDIA: Removable disk not in drive.
0x1051000B	STAT_BLANK: End of recorded data.
0x1051000C	STAT_BAD_CMD: Target reports "Illegal Request."
0x1051000D	STAT_NO_SENSE: Request sense returned no sense.

B

